🧭 Detailed Step-by-Step Roadmap (Beginner to Final Project)

**✅ Phase 0: Setup & Preparation (1–2 Days)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Set up your dev environment | Install [Python](https://www.python.org/downloads/), [VS Code](https://code.visualstudio.com/), and Git. |
| 2. Create a GitHub repo | Use git init, push your code regularly. |
| 3. Create project folder | Name it something like supply-chain-risk-analyzer. |

**📘 Phase 1: Understand the Project (2–3 Days)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Study “Supply Chain Risk” | Read [this summary](https://owasp.org/www-project-software-supply-chain-security/) or use YouTube. |
| 2. Understand dependencies | Learn what requirements.txt, package.json, and SBOM are. |
| 3. Research real-world attacks | Study Log4Shell, SolarWinds, Event-Stream, etc. |
| 4. Write a 1-page summary | Use this in your project report. |

**🧪 Phase 2: Learn Python & APIs (3–4 Days)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Python Basics | Learn: loops, lists, functions, file I/O, requests, json modules. Use [learnpython.org](https://www.learnpython.org/). |
| 2. API Basics | Learn how to send HTTP requests and handle responses. Use OSS Index or Safety API. |
| 3. Try This | Call a public API like https://api.agify.io/?name=tom and print response in Python. |

**📦 Phase 3: Build Dependency Reader (2 Days)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Parse requirements.txt | Use Python to read and extract dependencies. |
| 2. Validate packages | Filter out comments or invalid lines. |

**🔍 Phase 4: Vulnerability Checker (5–7 Days)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Sign up for OSS Index | <https://ossindex.sonatype.org/> |
| 2. Use requests to send packages | Convert list of packages to API-compatible format. |
| 3. Parse API response | Show CVEs, severities, and titles for each package. |
| 4. Create CLI output | Neatly print CVE info per package. |

**⚖️ Phase 5: Risk Scoring System (3–4 Days)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Design scoring model | e.g., Critical = 10, High = 7, Medium = 5, Low = 2 |
| 2. Sum package scores | Add scores to get total risk score per project. |
| 3. Categorize risk | (Low, Medium, High, Critical) based on total score. |

📄 Phase 6: Report Generator (Optional Web or CLI) (1 Week)

| **Option** | **Tool** | **What to Do** |
| --- | --- | --- |
| Simple | CLI | Print results to terminal or save .txt/.csv file |
| Advanced | Flask or Streamlit | Create a web form that uploads requirements.txt and shows results |
| Bonus | Charts | Add pie/bar charts for vulnerability count using matplotlib or Chart.js |

**🧾 Phase 7: SBOM Support (Optional, Advanced)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Learn about SBOM | Use [CycloneDX](https://cyclonedx.org/) or SPDX formats |
| 2. Export packages list as JSON | Generate mini SBOM from requirements.txt |
| 3. (Optional) Compare against public SBOM tools |  |

**📑 Phase 8: Final Report, README, and Presentation (1–2 Weeks)**

| **Task** | **What to Do** |
| --- | --- |
| 1. Write Final Report | Include intro, tools, method, screenshots, results, references. I can help. |
| 2. Write README | Project description, install instructions, usage example. |
| 3. Create PowerPoint | For final year demo or job interviews. I can help design this. |

| **Term** | **Description** |
| --- | --- |
| **Repository (Repo)** | Project folder on GitHub |
| **Clone** | Copying a repo to your local machine |
| **Fork** | Copy of a repo to your own GitHub account for experimentation |
| **Commit** | Saving a snapshot of your changes |
| **Branch** | A parallel version of the repo to work on features independently |
| **Pull Request (PR)** | Suggesting changes from your branch to the main branch |
| **Merge** | Combining code from different branches |
| **Issues** | Used to track bugs, features, or tasks |

**Research**

**Definitions and Key Concepts**

* **Software Supply Chain (SSC):** The full set of tools, code, dependencies, and steps used to develop and deliver software. This includes IDEs, VCS (Version Control Systems), third-party libraries, CI/CD pipelines, and configuration tools.
* **Supply Chain Risk:** The possibility that any part of the software supply chain could be attacked or compromised, leading to vulnerabilities in the final product.
* **Software Supply Chain Security (SSCS):** Measures taken to protect all components in the SSC from being exploited.

**🔹 Types of Supply Chain Attacks**

1. **Source Code Threats:**
   * Injecting malicious or vulnerable code
   * Exploiting VCS systems or unauthorized code changes
2. **Build Environment Threats:**
   * Compromising the build system or tools
   * Using untrusted build sources
3. **Dependency Threats:**
   * Using vulnerable or compromised third-party software (both direct and transitive dependencies)
4. **Deployment & Runtime Threats:**
   * Misconfigured CI/CD pipelines
   * Privilege misuse
   * Compromised binaries in runtime

**🔹 Real-World Examples & Impacts**

* **SolarWinds (2020):** Attackers inserted malware into a software update, affecting thousands of customers including governments.
* **Codecov (2021):** A malicious script in their CI/CD pipeline leaked credentials and sensitive data from clients.

**Impact:**

* Widespread compromise across many downstream users.
* Loss of confidentiality, integrity, and availability.
* Financial loss, espionage, and trust damage.

**🔹 Prevention & Mitigation Techniques**

**✅ General Practices**

* Enforce **strong access control**: MFA, least privilege, no plain-text credentials.
* Enable **logging & monitoring**: Use centralized logging systems (e.g., SIEM).
* Use **security automation tools**: SAST, DAST, SCA, container scanners.

**✅ Source Code Security**

* Conduct **peer reviews** before merging code.
* Secure VCS with branch protections and audit tools like **Legitify**.
* Secure developer environments with endpoint security.

**✅ Dependency Management**

* **Assess suppliers** (open-source and proprietary).
* Maintain **SBOMs** (Software Bill of Materials) to track dependencies.
* Monitor dependencies using tools like **OWASP Dependency Check**, **retire.js**.
* Use **lockfiles** to pin versions and prevent unwanted updates.

**✅ Build Environment Security**

* **Inventory and harden** all build tools and infrastructure.
* Use **code signing** and verify signatures.
* Store and manage build scripts/configs in source control.
* Generate **provenance metadata** to verify the source and integrity of artifacts.
* Use **ephemeral build environments** (e.g., containers) for isolation.
* Avoid excessive use of user-defined build parameters.

**✅ Deployment & Runtime Protections**

* Scan final binaries for secrets or anomalies.
* Continuously monitor deployed software and infrastructure for vulnerabilities or misconfigurations.