**🛠️ AWS Blast Radius Simulator – Development Roadmap**

| **Phase** | **Goal** | **Key Tasks** | **Deliverables** | **Time Estimate** |
| --- | --- | --- | --- | --- |
| **1. Core Engine (MVP)** | Simulate privilege escalation paths from static IAM policy | - Parse IAM policy JSON - Identify risky IAM actions (e.g., PassRole, AssumeRole, UpdateFunctionCode) - Build access graph using NetworkX - Map service-to-service lateral flow | - Working CLI prototype - Graph object of access paths - Text summary of blast radius | 1.5 – 2 weeks |
| **2. Visualization Layer** | Make blast radius tangible via visual graphs | - Integrate Mermaid.js or Graphviz - Map nodes (IAM, services) + privilege levels - Export graph as HTML or Markdown | - Visual attack path output - Color-coded graph diagram - Simple exportable report | 1 week |
| **3. Lightweight Web UI** | Build basic front end to improve usability | - Flask app setup - Upload/parse IAM policy - Render graph in browser - Output attack summary and scoring (text + visual) | - Hosted local web demo - Clean UI for policy upload - Real-time graph rendering | 1 – 1.5 weeks |
| **4. Risk Scoring Engine** | Prioritize identities based on potential damage | - Build scoring logic (e.g., service sensitivity × access level × escalation potential) - Display score in UI - Update graph with risk-weighted paths | - Risk score for each identity - Weighted visual graph - Downloadable risk report | 1 – 1.5 weeks |
| **5. Real IAM Integration** | Enable fetching IAM roles/policies from live AWS | - Use Boto3 to fetch attached/inline policies - Integrate IAM Access Analyzer (if needed) - Parse live data into simulator | - Dynamic IAM entity support - AWS account integration - CLI or UI policy loader from AWS | 2 weeks |
| **6. Polish & Launch** | Prep for open-source, visibility, and sharing | - Write clear README & docs - Add sample IAM policies - Record demo / write blog - Launch on GitHub + share on LinkedIn | - Public GitHub repo - Demo content & walkthrough - Community-ready project |  |

**✅ Phase 1 – Task Breakdown Table**

| **Task** | **What to Do** | **Tools/Notes** |
| --- | --- | --- |
| **1. Accept a JSON IAM Policy as Input** | Create a script (simulate.py) that takes a local policy file (policy.json). This should be a PolicyDocument JSON structure from AWS. | Use argparse for CLI input |
| **2. Parse the Policy Document** | Extract all Action, Effect, and Resource entries from the JSON. Structure it into a list of effective permissions. | Python’s json module |
| **3. Normalize Wildcards & Service-Actions** | Normalize actions like s3:\* into full actions (e.g., s3:GetObject, s3:PutObject). Either expand manually or reference a complete AWS actions list. | Use AWS Actions JSON (can be scraped or sourced from GitHub/AWS docs) Regex for wildcards |
| **4. Identify Privilege Escalation Vectors** | Match allowed actions against known escalation methods (e.g., iam:PassRole, sts:AssumeRole, lambda:UpdateFunctionCode). Create simple if/else rules or tag actions as “escalation-risk”. | Use Rhino Security Labs matrix Custom escalation\_rules.py lookup file |
| **5. Build Access Graph Representation** | Create a directed graph using NetworkX: Nodes = AWS services or identity types Edges = Allowed access paths Add attributes like access type (read/write/admin), escalation, etc. | networkx.DiGraph() Use node/edge attributes to store metadata |
| **6. Output Graph to Console** | For MVP, just print out reachable services and escalation paths to terminal. Optionally export the graph in .dot or .graphml format. | Use networkx.write\_dot() or pydot Optional: print adjacency list or path trace |
| **7. Generate Basic Blast Radius Summary** | Output a summary like: “X services accessible” “Y privilege escalation paths detected” “Z critical services involved” | Plaintext output to terminal or save to .txt Highlight escalation and high-risk services |