Project Title: Autonomous Network Attack Planner Using Reinforcement Learning

---

Short Project Description:

AI-driven system that autonomously plans and simulates network attack strategies using reinforcement learning (RL). The tool maps target networks, generates adaptive attack paths, and logs results for security analysis and defense hardening.

Component	Role
Network Topology Mapper	Builds a live map of the target network
Action Space Generator	Defines possible attacker actions (scan, exploit, pivot, etc)
RL Attack Planner	Uses reinforcement learning to plan attack paths
Execution Engine	Carries out chosen actions safely
Attack Path Logger	Logs steps and results for reporting

Component Details:

1. **Network Topology Mapper**:
   - Scans the network (Nmap, SNMP walking, etc).
2. **Action Space Generator**:
   - Defines what actions are possible at each point:
     - Port scanning
     - Vulnerability scanning
     - Exploitation
     - Privilege escalation
     - Lateral movement
     - Etc
3. **RL Attack Planner**:
   - Trains a reinforcement learning agent:
     - Rewards based on success (e.g., gaining shells, moving laterally, etc).
   - Learns optimal attack strategies dynamically.
4. **Execution Engine**:
   - Safely executes planned actions (simulated or real, depending on test settings).
5. **Attack Path Logger**:
   - Tracks all actions, rewards, states, and results.

Overall System Flow:

- Input: Target network IP range
- Output: Attack graph and sequence of actions taken
- Focus on **adaptive, intelligent attack strategy generation**.

Internal Functioning of Each Module:

1. Network Topology Mapper

- **Scanning**:
  - Uses Nmap, SNMP walk, NetBIOS scanning, etc.
- **Graph Building**:
  - Nodes: Devices/hosts
  - Edges: Connections (ports open, trust relationships, etc)
- **Stored as**:
  - Graph data structures (NetworkX, Graph-tool libraries, etc)

---

2. Action Space Generator

- **Action Types**:
  - Port Scan
  - Vulnerability Scan
  - Exploit Service
  - Brute-force credentials
  - Move Laterally
  - Etc
- **Dynamic Expansion**:
  - As new hosts are discovered, action space expands.
- **Action Representation**:
  - Encoded as vectors for RL models.

---

3. RL Attack Planner

- **Algorithm Choices**:
  - Q-Learning
  - Deep Q-Networks (DQN)
  - Policy Gradient Methods (PPO, A3C)
  - Etc
- **Reward Signal (e.g.)**:
  - +10 for discovering new host
  - +20 for successful exploit
  - +50 for achieving domain admin
  - -5 for failed exploit (to discourage noise)

- **State Representation**:
    - Current network graph state
    - Known services and vulnerabilities

---

4. Execution Engine

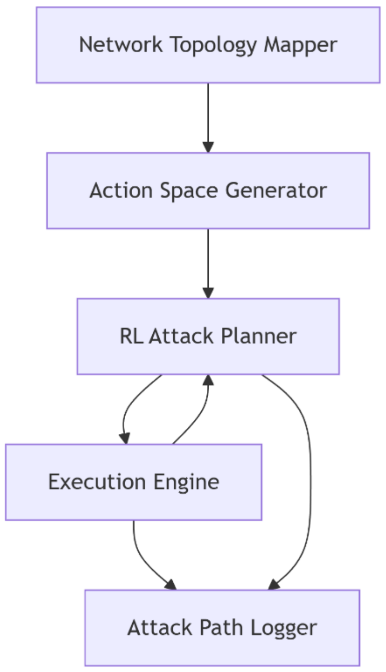
- **Safe Mode**:
    - Executes simulated actions first (emulated targets).
- **Real Mode (optional)**:
    - Launches real exploits only if authorized.

---

5. Attack Path Logger

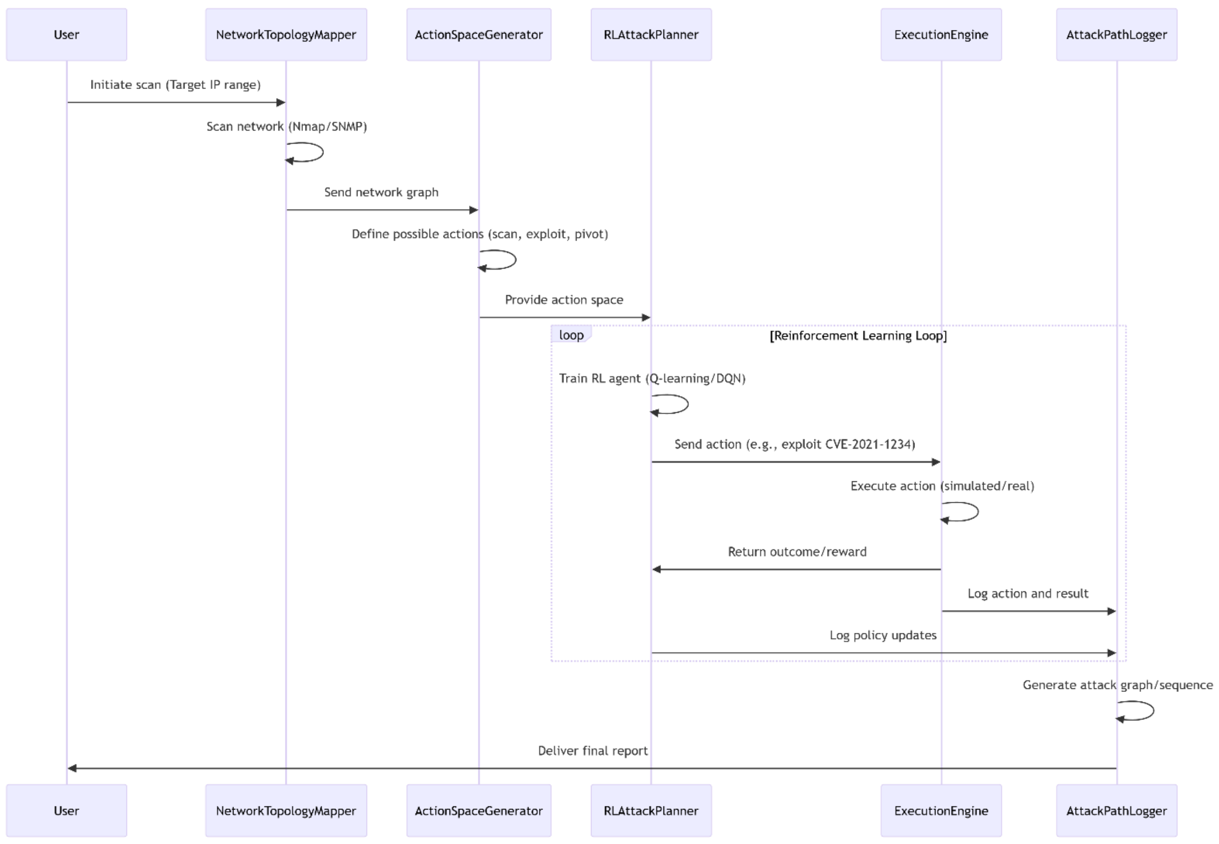
- **Logs**:
    - Time
    - Action
    - Target
    - Outcome
    - Reward
    - Etc
- **Used for**:
    - Replay analysis
    - Attack visualization graphs

Component Diagram



- The **Network Topology Mapper** builds the network graph and feeds it to the **Action Space Generator**.
- The **Action Space Generator** defines possible attacker actions (e.g., exploit, pivot, etc) and sends them to the **RL Attack Planner**.
- The **RL Attack Planner** selects optimal actions using reinforcement learning and dispatches them to the **Execution Engine**.
- The **Execution Engine** executes actions, provides feedback to the RL model for learning, and logs results in the **Attack Path Logger**.
- The **Attack Path Logger** aggregates data from both the Execution Engine and RL Planner to generate reports.

Sequence Diagram


User
NetworkTopologyMapper
ActionSpaceGenerator
RLAttackPlanner
ExecutionEngine
AttackPathLogger
Initiate scan (Target IP range)
Scan network (Nmap/SNMP)
Send network graph
Define possible actions (scan, exploit, pivot)
Provide action space
loop
[Reinforcement Learning Loop]
Train RL agent (Q-learning/DQN)
Send action (e.g., exploit CVE-2021-1234)
Execute action (simulated/real)
Return outcome/reward
Log action and result
Log policy updates
Generate attack graph/sequence
Deliver final report

- The **User** triggers the process by specifying a target network.
- The **Network Topology Mapper** scans the network and sends the graph to the **Action Space Generator**.

- The **RL Attack Planner** iteratively trains its model by executing actions via the **Execution Engine**, receiving rewards, and refining strategies.
- Actions and outcomes are logged in real-time by the **Attack Path Logger**.
- After completing the attack simulation, the **Attack Path Logger** compiles the results into an actionable report for the **User**.

Detailed Project Description: Autonomous Network Attack Planner Using Reinforcement Learning

An AI-driven system that autonomously plans and simulates network attack strategies using reinforcement learning (RL). The system maps target networks, generates adaptive attack paths, and logs results for security analysis and defense hardening.

---

1. System Overview

The system intelligently simulates or executes network attacks (e.g., exploitation, lateral movement, etc) using RL to optimize strategies. It focuses on **adaptive planning**, **safe execution**, and **actionable reporting** for red teaming or penetration testing.

Key Components

1. **Network Topology Mapper**
2. **Action Space Generator**
3. **RL Attack Planner**
4. **Execution Engine**
5. **Attack Path Logger**

---

2. Component Design & Implementation

2.1 Network Topology Mapper

Functionality:

- Discovers and maps the target network’s devices, services, and connections.

Implementation Steps (e.g.):

1. **Scanning Tools**:
    - **Nmap**: For port scanning (`nmap -sV -O <target>`).

    - **SNMP Walk**: Enumerate SNMP-enabled devices (`snmpwalk -v2c -c public <IP>`).
    - **ARP Scanning**: Identify live hosts (`arp-scan -l`).
    - **Etc**.
2. **Graph Construction**:
    - Use `NetworkX` to model the network as a graph:
        - **Nodes**: Hosts (IP, OS, services).
        - **Edges**: Open ports, protocols, trust relationships, etc.
3. **Output**:
    - JSON/GraphML file representing the network topology.

2.2 Action Space Generator

Functionality:

- Defines possible attacker actions based on the network state.

Implementation Steps (e.g.):

1. **Action Types**:
    - **Reconnaissance**: Port scan, service detection, etc.
    - **Exploitation**: Use known vulnerabilities (CVE-2023-XXXX), etc.
    - **Lateral Movement**: SSH brute-forcing, pass-the-hash, etc.
    - **Privilege Escalation**: Exploit misconfigured sudo rights, etc.
    - **Etc**
2. **Dynamic Expansion**:
    - Update actions as new hosts/services are discovered (e.g., add "exploit Apache 2.4.49" if detected).
3. **Encoding Actions**:
    - Represent actions as vectors (e.g., `[action_type, target_ip, port]`).

Output:

- List of valid actions for the RL agent.

2.3 RL Attack Planner

Functionality:

- Trains an RL agent to select optimal attack sequences.

Implementation Steps (e.g.):

1. **Algorithm Selection**:
    - **Deep Q-Networks (DQN)**: For discrete action spaces.
    - **Proximal Policy Optimization (PPO)**: For continuous or hybrid action spaces.
    - **Etc**.
2. **Reward Design**:
    - **Positive Rewards (e.g.)**:
        - `+10` for discovering a new host.
        - `+20` for successful exploitation.
        - `+50` for achieving domain admin.
    - **Negative Rewards**:
        - `-5` for failed exploit attempts.
3. **State Representation**:
    - Encode the network graph, known vulnerabilities, and access levels.
4. **Training Loop**:
    - Use `TensorFlow` or `Stable Baselines3` to train the RL model.

Output:

- Optimized attack policy for the target network.

2.4 Execution Engine

Functionality:

- Executes planned actions safely (simulated or real).

Implementation Steps (e.g.):

1. **Simulation Mode**:
   - Use emulated targets (e.g., Docker containers with vulnerable services).
2. **Real Execution**:
   - Integrate tools like **Metasploit** or **Hydra** for exploitation.
   - Ensure authorization and ethical guidelines (e.g., scope of testing).
3. **Safety Measures**:
   - Rate limiting to avoid detection.
   - Rollback mechanisms (e.g., snapshot restoration).

Output:

- Action outcomes (success/failure) and new network states.

2.5 Attack Path Logger

Functionality:

- Logs attack steps and generates reports.

Implementation Steps (e.g.):

1. **Log Structure**:
   - Fields: Timestamp, action type, target, outcome, reward.
   - Example:

     ```
     {
       "timestamp": "2023-10-05T14:30:00Z",
       "action": "exploit_apache_cve2023",
       "target": "192.168.1.10:80",
       "outcome": "success",
     ```

```
  "reward": 20
}
```

2. **Visualization**:
    - Use `Gephi` or `Cytoscape` to render attack graphs.
3. **Reporting**:
    - Generate HTML/PDF reports with timelines and attack paths.

Output:

- Interactive attack graph and structured report.

---

3. Technology Stack (e.g.)

- **Network Scanning**: Nmap, SNMPWalk, ARP-Scan, etc.
- **RL Frameworks**: TensorFlow, Stable Baselines3, Ray RLlib, etc.
- **Exploitation Tools**: Metasploit, Hydra, SQLMap, etc.
- **Visualization**: NetworkX, Gephi, Grafana, etc.

---

4. Evaluation & Validation

1. **Effectiveness Metrics**:
    - **Success Rate**: Percentage of objectives achieved (e.g., domain compromise).
    - **Time-to-Compromise**: Average time to reach critical targets.
    - **Etc**.
2. **Baseline Comparison**:
    - Compare against manual penetration testing or scripted attacks.
3. **Safety Testing**:
    - Ensure no unintended impact in simulation mode.

---

5. Development Roadmap

1. **Phase 1**: Build Network Topology Mapper and Action Space Generator.
2. **Phase 2**: Train RL models and integrate Execution Engine.
3. **Phase 3**: Implement Attack Path Logger and reporting.
4. **Phase 4**: Validate on lab networks (e.g., Hack The Box).

---

6. Challenges & Mitigations (optional)

- **Scalability**: Use parallel scanning and distributed RL training.
- **Stealth**: Implement delays and randomize scan patterns to evade IDS.
- **Model Convergence**: Use curriculum learning (start with simple networks).

---

7. Glossary

- **RL**: Reinforcement Learning
- **DQN**: Deep Q-Network
- **PPO**: Proximal Policy Optimization
- **CVE**: Common Vulnerabilities and Exposures
