Project Title: Automatic Evasion Path Discovery in Layered Firewall Architectures

Core Idea:

- Create a system that **automatically maps layered firewalls** (perimeter, DMZ, internal segmentation) and **finds viable evasion paths** through multi-firewall environments.

Component	**Role**
Layered Firewall Mapper	Discovers existence and rough layout of multiple firewalls
Rule Behavior Inference Engine	Probes firewall rules via intelligent traffic patterns
Evasion Path Planner	Plans sequences of protocol abuses, tunneling, etc.
Dynamic Testing Engine	Executes evasion attempts against real network
Visualization Module	Graphs firewall architectures and successful bypass paths

Component Details:

1. **Layered Firewall Mapper**:
   - Techniques:
     - TTL hopping analysis.
     - MTU-based path detection.
     - Timing analysis.
     - Etc.
2. **Rule Behavior Inference Engine**:
   - Send:
     - Varied packets (different ports, payload types),
     - Watch accept/deny responses.
   - Infer rule sets:
     - Protocol whitelists,
     - Port-specific behaviors,
     - Etc.
3. **Evasion Path Planner**:
   - Plans:
     - VPN chaining,
     - HTTP tunneling,
     - DNS tunneling,
     - Etc.
4. **Dynamic Testing Engine**:
   - Launches crafted evasion flows.
5. **Visualization Module**:
   - Produces:

- Graphs of firewall locations,
- Rules inferred,
- Paths through them,
- Etc.

Overall System Flow:

- Input: Targeted multi-firewall environment
- Output: Mapped firewall rules + possible evasion strategies
- Focus: **Intelligent multi-layered firewall penetration**

Internal Functioning of Each Module:

1. Layered Firewall Mapper

- **Discovery techniques**:
  - **TTL Analysis**:
    - Increment TTL, observe ICMP "time exceeded" messages.
    - Map hop counts to discover layered firewalls.
  - **Port-based Segmentation**:
    - Try different ports to infer different firewall behaviors per layer.
  - Etc

---

2. Rule Behavior Inference Engine

- **Probe tactics**:
  - Send variations:
    - Payload size,
    - Protocol (ICMP/TCP/UDP),
    - Application type (HTTP/SSH/FTP),
    - Etc.
- **Inference model**:
  - Build:
    - Allowed / Blocked rule maps per firewall layer.

---

3. Evasion Path Planner

- **Path building**:
  - Plan:
    - Tunnels (HTTP Connect, DNS tunneling),
    - Obfuscations (TLS handshake fragmentation),
    - Chained proxies,

- Etc.

---

4. Dynamic Testing Engine

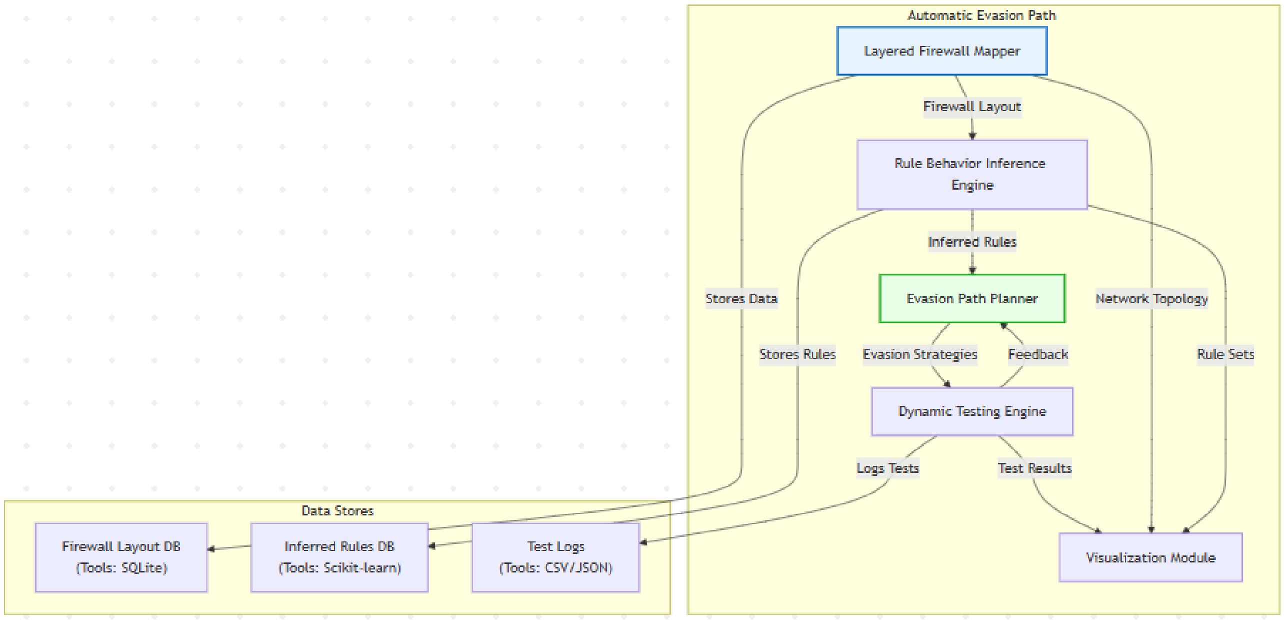
- **Execution**:
  - Actively test paths,
  - Adjust if responses change dynamically.

---

5. Visualization Module

- **Graph outputs**:
  - Nodes: Firewall stages.
  - Edges: Successful/failed evasion paths.

Component Diagram



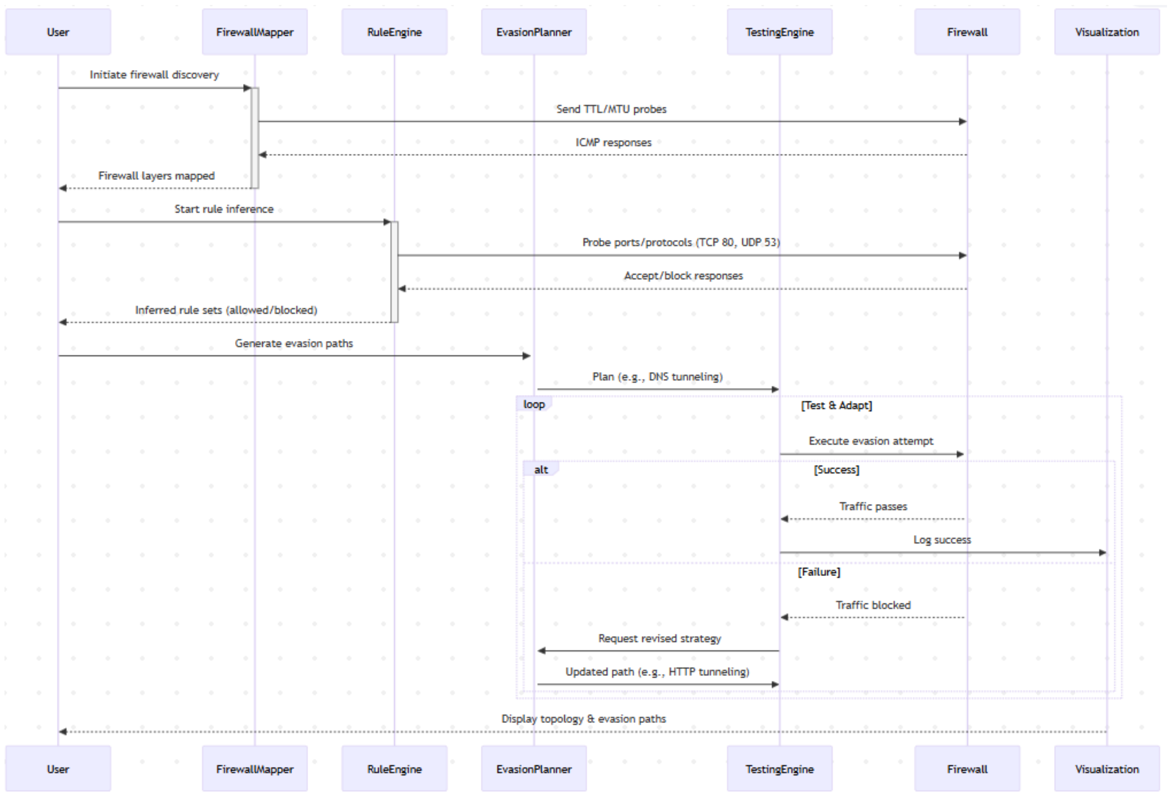
Explanation:

1. **Components**:
   - **Layered Firewall Mapper**: Discovers firewall layers using TTL/MTU analysis, timing probes, etc. Outputs network topology.
   - **Rule Behavior Inference Engine**: Probes firewalls to infer allowed/blocked rules (e.g., ports, protocols, etc). Stores rules in a database.

   - **Evasion Path Planner**: Generates bypass strategies (e.g., DNS tunneling, HTTP tunneling, etc) based on inferred rules.
   - **Dynamic Testing Engine**: Executes evasion paths and logs results. Provides feedback to refine strategies.
   - **Visualization Module**: Displays firewall topology, rule heatmaps, and successful evasion paths.

2. **Data Stores**:
   - **Firewall Layout DB**: Stores network topology and firewall layer data (e.g., hop counts, MTU points, etc).
   - **Inferred Rules DB**: Contains allowed/blocked port/protocol rules per firewall layer.
   - **Test Logs**: Records evasion attempts (success/failure) for analysis.
3. **Workflow**:
   - **Firewall Mapper → Rule Engine**: Shares firewall layout for rule probing.
   - **Rule Engine → Evasion Planner**: Provides inferred rules to plan bypass tactics.
   - **Evasion Planner → Testing Engine**: Sends strategies for execution.
   - **Testing Engine → Visualization**: Logs results for graphical display.
   - **Feedback Loop**: Testing results refine evasion strategies iteratively.
4. **Key Features**:
   - **Topology Discovery**: Uses TTL/MTU analysis to map layered firewalls.
   - **Rule Inference**: Machine learning (Scikit-learn) to classify allowed/blocked traffic.
   - **Dynamic Testing**: Ansible/asyncio for automated evasion testing and adaptation.
   - **Interactive Visualization**: Plotly/Graphviz dashboards to display paths and rules.

Sequence Diagram



Explanation:

1. **Firewall Discovery**:
   - The **User** initiates the **Firewall Mapper**, which sends TTL/MTU probes to map layered firewalls.
   - The **Firewall** responds with ICMP messages, allowing the mapper to build the network topology.
2. **Rule Inference**:
   - The **Rule Engine** probes firewall ports/protocols (e.g., TCP port 80, UDP port 53) to infer allowed/blocked rules.
3. **Evasion Planning & Testing**:
   - The **Evasion Planner** generates bypass strategies (e.g., DNS tunneling).
   - The **Testing Engine** executes the plan. If blocked, it triggers a feedback loop to revise strategies (e.g., switch to HTTP tunneling).
4. **Visualization**:

- Successful paths and firewall layers are visualized for the user, showing viable evasion routes.

Detailed Project Description: Automatic Evasion Path Discovery in Layered Firewall Architectures

This system automates the discovery of evasion paths through multi-layered firewall environments (e.g., perimeter, DMZ, internal segmentation). By mapping firewall layers, inferring rules, and testing bypass strategies, the system identifies stealthy paths for penetration testing or red teaming.

---

1. Core Components & Implementation Details

1.1 Layered Firewall Mapper

- **Role**: Discover firewall layers and network topology.
- **Implementation (e.g.)**:
    - **TTL Analysis**:
        - Send ICMP/UDP packets with incrementing TTL values.
        - Analyze ICMP "Time Exceeded" messages to map hops and identify firewall boundaries.
        - Etc.
    - **MTU Detection**:
        - Send large packets (e.g., 1500+ bytes) to detect fragmentation points.
        - Etc.
    - **Timing Analysis**:
        - Measure latency spikes to infer stateful inspection points.
        - Etc.
    - Etc
- **Tools (e.g.)**: Scapy (packet crafting), Python’s `ping3` library, etc.

1.2 Rule Behavior Inference Engine

- **Role**: Probe and infer firewall rules (allowed/blocked protocols, ports).
- **Implementation (e.g.)**:
    - **Active Probing**:

    - Craft packets targeting specific ports (e.g., 22, 80, 443, etc) and protocols (TCP/UDP/ICMP).
    - Use `nmap`-like techniques to detect open/closed ports.
  - **Rule Inference**:
    - Build a ruleset database using decision trees or logistic regression.
  - Etc.
- **Tools (e.g.)**: Scapy, Scikit-learn (for rule classification), SQLite (for rule storage), etc.

1.3 Evasion Path Planner

- **Role**: Generate bypass strategies based on inferred rules.
- **Implementation (e.g.)**:
  - **Tunneling Methods**:
    - **HTTP Tunneling**: Encode traffic in HTTP headers (e.g., `Cookie` fields).
    - **DNS Tunneling**: Use DNS queries/responses to exfiltrate data.
    - **VPN Chaining**: Route traffic through multiple VPN gateways.
    - **Etc**.
  - **Obfuscation Tactics**:
    - Fragment TLS handshakes or use non-standard ports (e.g., HTTPS over port 8080).
    - Etc
- **Tools (e.g.)**: `dnscat2` (DNS tunneling), `stunnel` (SSL/TLS wrapping), etc.

1.4 Dynamic Testing Engine

- **Role**: Execute and refine evasion paths.
- **Implementation (e.g.)**:
  - **Automated Testing**:
    - Script evasion attempts using Python’s `subprocess` or Ansible.
    - Handle retries and fallback strategies (e.g., switch to DNS if HTTP fails).
  - **Feedback Loop**:
    - Log successes/failures and update the Evasion Path Planner.
- **Tools (e.g.)**: Ansible (playbooks), Python’s `asyncio` (parallel testing), etc.

1.5 Visualization Module

- **Role**: Graph firewall layers and successful paths.
- **Implementation (e.g.)**:
  - **Network Graphs**:
    - Use `networkx` to model nodes (firewalls) and edges (evasion paths).
    - Highlight paths with `matplotlib` or interactive tools like Plotly.
  - **Rule Heatmaps**:
    - Visualize allowed/blocked ports and protocols per layer.
- **Tools (e.g.)**: Plotly Dash (interactive dashboards), Graphviz (topology diagrams).

---

2. System Workflow

1. **Firewall Discovery**: TTL/MTU analysis identifies layered firewalls.
2. **Rule Probing**: Send crafted packets to infer allow/block rules.
3. **Path Planning**: Generate evasion tactics (e.g., DNS tunneling, etc).
4. **Dynamic Testing**: Execute and adjust strategies based on feedback.
5. **Visualization**: Display firewall architecture and successful paths.

---

3. Evaluation Metrics

- **Evasion Success Rate**: % of bypassed firewall layers.
- **Rule Inference Accuracy**: Precision/recall of inferred rules.
- **Time-to-Path**: Duration to identify a viable evasion path.

---

4. Tools & Frameworks (e.g)

- **Network Analysis**: Scapy, Wireshark, nmap, etc.
- **Machine Learning**: Scikit-learn, TensorFlow (for advanced rule inference), etc.

- **Visualization**: Plotly, Graphviz, NetworkX, etc.
- **Automation**: Ansible, Python's `asyncio, etc`.

---

5. Suggested Implementation Steps (e.g.)

1. **Setup Test Environment**:
   - Deploy layered firewalls (e.g., pfSense, iptables, etc).
   - Install dependencies: `pip install scapy plotly networkx`.
2. **Implement Firewall Mapper**:
   - Write Python scripts for TTL/MTU analysis.
   - Example: Use Scapy to send ICMP packets with TTL=1,2,3... and log responses.
3. **Build Rule Inference Engine**:
   - Craft TCP SYN packets to probe ports 1-1024.
   - Train a classifier to map allowed/blocked rules.
4. **Develop Evasion Planner**:
   - Integrate tunneling tools (e.g., `dnscat2` for DNS tunneling).
   - Generate evasion paths as JSON workflows (e.g., `{"method": "HTTP", "port": 80}`).
5. **Automate Testing**:
   - Use Ansible playbooks to execute evasion paths.
   - Log results in CSV/JSON for feedback.
6. **Create Visualization Dashboard**:
   - Use Plotly Dash to build an interactive map of firewall layers.
   - Color-code nodes by rule strictness (red=strict, green=permissive).

---

6. Challenges & Mitigations (optional)

- **Dynamic Firewalls**: Periodically re-probe rules and update paths.
- **Detection Risks**: Randomize probe timing and mimic legitimate traffic patterns.

- **Protocol Compliance**: Validate traffic with tools like `tcpreplay` or `ostinato`.
