Project Title: Self-Healing Infrastructure through LLM-driven Vulnerability Mitigation

---

Core Idea:

- Combine **dynamic attack detection** + **automatic Gen-AI patching**.
- Systems fix themselves without waiting for human action.

Component	Role
Continuous Risk Monitor	Detects vulnerabilities, config issues in live systems
LLM Patch Suggester	Drafts configuration or code fixes dynamically
Validation Sandbox	Runs auto-patches in test environment first
Patch Deployment Engine	Applies safe patches in production
Incident Response Generator	Documents auto-healing actions for audit trails

---

Component Details:

1. **Continuous Risk Monitor**:
    - Monitors:
        - CVE databases
        - Dependency scanning
        - Infrastructure misconfigurations
        - Etc
2. **LLM Patch Suggester**:
    - Given a risk:
        - Proposes configuration tweaks, version upgrades, permission fixes, etc.
    - Understands app stack and dependencies.
3. **Validation Sandbox**:
    - Runs patches on cloned environments to verify no breakage.
4. **Patch Deployment Engine**:
    - Pushes patches to production if safe.
5. **Incident Response Generator**:
    - Creates timestamped, human-readable reports of actions taken.

Overall System Flow:

- Input: Continuous vulnerability and misconfiguration feeds
- Output: Self-repairing secure systems
- Focus: **Real-time automatic vulnerability remediation**

Internal Functioning of Each Module:

1. Continuous Risk Monitor

- **Monitoring**:
  - Subscribes to:
    - CVE feeds (NVD, vendor advisories, etc)
    - Dependency checkers (Snyk, Dependabot, etc)
    - Cloud misconfiguration scanners (ScoutSuite, Prowler, etc)
    - Etc.

---

2. LLM Patch Suggester

- **Generation**:
  - LLM reads vulnerability reports.
  - Suggests:
    - Config patches (e.g., firewall rules, etc)
    - Library upgrades
    - Software upgrades
    - Permission tightening
    - Etc
- **Examples**:
  - `"Upgrade OpenSSL from 1.0.1 to 1.1.1."`
  - `"Add HTTP security headers to nginx.conf."`

---

3. Validation Sandbox

- **Safety testing**:
  - Clones production environments.
  - Applies patches automatically.
  - Runs:
    - Unit tests
    - Integration tests
    - Fuzz tests
    - Etc

---

4. Patch Deployment Engine

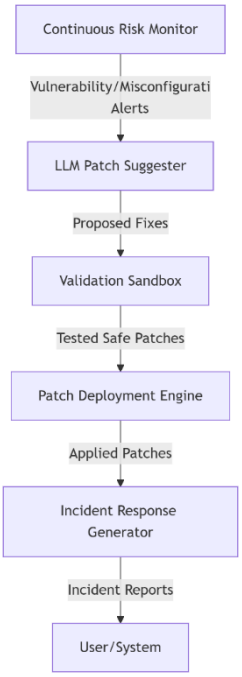
- **Deployment**:
  - If tests pass, push to production.
  - Methods:
    - Blue/green deployment
    - Rolling updates

---

5. Incident Response Generator

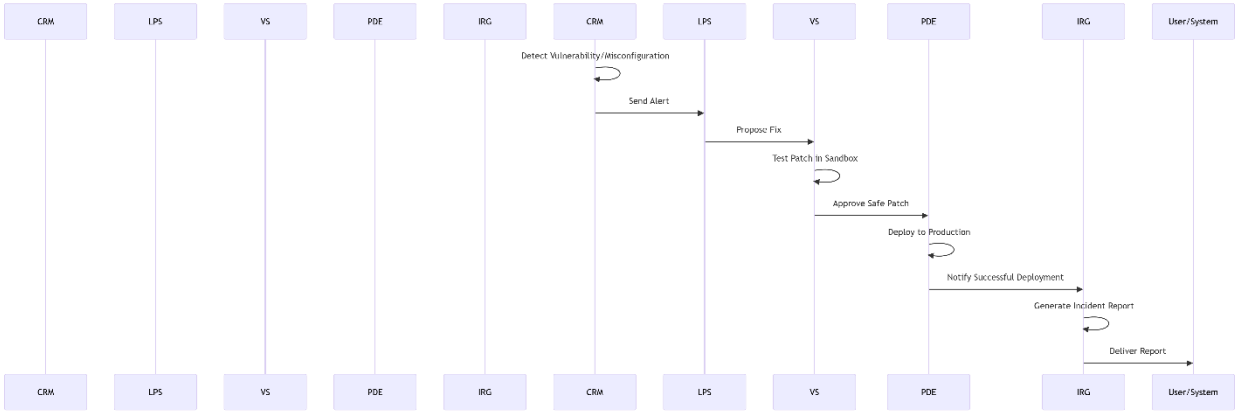
- **Documentation**:
    - Creates structured incident reports:
        - Time of patch
        - Detected issue
        - Auto-mitigation taken
        - Residual risks

Component Diagram



- **Continuous Risk Monitor**: Detects vulnerabilities (CVEs), dependency issues, and misconfigurations in real time.
- **LLM Patch Suggester**: Uses LLMs to generate code/config fixes (e.g., library upgrades, firewall rules, etc).
- **Validation Sandbox**: Tests patches in a cloned environment to ensure stability.
- **Patch Deployment Engine**: Safely deploys validated fixes to production (e.g., blue/green deployment, etc).
- **Incident Response Generator**: Documents automated actions (patches applied, test results, residual risks, etc).

Sequence Diagram



1. **Continuous Risk Monitor** detects a vulnerability (e.g., outdated OpenSSL, etc).
2. **LLM Patch Suggester** drafts a fix (e.g., "Upgrade OpenSSL to 1.1.1").
3. **Validation Sandbox** tests the patch via unit/integration tests.
4. **Patch Deployment Engine** applies the patch to production if tests pass.
5. **Incident Response Generator** creates a report and delivers it to the **User/System**.

Detailed Project Description: Self-Healing Infrastructure through LLM-driven Vulnerability Mitigation

An autonomous system that detects vulnerabilities, generates AI-driven fixes, tests and deploys patches, and documents remediation actions. This system minimizes human intervention while ensuring infrastructure resilience.

---

1. System Components and Roles

1.1 Continuous Risk Monitor

Purpose: Continuously scan for vulnerabilities, misconfigurations, and outdated dependencies.
Implementation Details (e.g.):

- **Tools**:
    - **Dependency Scanners**: Snyk, Dependabot (for code libraries), etc.
    - **Cloud Security**: ScoutSuite, Prowler (AWS/Azure/GCP misconfigurations), etc.
    - **CVE Feeds**: NVD API, OSV Database, etc.
    - Etc.
- **Integration**:
  ```
  import requests
  nvd_response = requests.get("https://services.nvd.nist.gov/rest/json/cves/1.0?keyword=OpenSSL")
  cvss_score = nvd_response.json()["result"]["CVE_Items"][0]["impact"]["baseMetricV3"]["cvssV3"]["baseScore"]
  ```
- **Alerting**: Slack/Email notifications via webhooks for critical CVEs.

1.2 LLM Patch Suggester

Purpose: Generate code/config fixes using Large Language Models.
Implementation Details (e.g.):

- **LLM Prompts**:
  ```
  prompt = f"""
  Vulnerability: OpenSSL 1.0.1 is vulnerable to Heartbleed (CVE-2014-0160).
  Current environment: Ubuntu 22.04, Apache 2.4.57.
  ```

```
Suggest a fix.
"""
response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "user", "content": prompt}]
)
# Output: "Upgrade OpenSSL to 1.1.1 and recompile Apache."
```

- **Fine-Tuning**: Train LLMs on patch commit histories (e.g., GitHub repositories, etc) for context-aware suggestions.

1.3 Validation Sandbox

Purpose: Test patches in an isolated environment to prevent production failures.
Implementation Details (e.g.):

- **Cloning Infrastructure**:
  - Use Terraform to replicate cloud environments.
  - Docker Compose for local service stacks.
  - Etc.
- **Automated Testing (yaml)**:

```
# GitHub Actions Example
jobs:
  validate-patch:
    runs-on: ubuntu-latest
    steps:
      - name: Run unit tests
        run: pytest tests/
      - name: Fuzz test API
        run: ./fuzz_target_api.py
```

1.4 Patch Deployment Engine

Purpose: Safely deploy validated patches to production.
Implementation Details (e.g.):

- **Strategies**:
  - **Blue/Green Deployment**:

    ```
    kubectl apply -f green-deployment.yaml
    kubectl switch traffic green
    ```

  - **Rolling Updates**:

    ```
    kubectl rollout restart deployment/nginx
    ```

- **Rollback Mechanism**:
```
if integration_tests_fail:
    kubectl rollout undo deployment/nginx
```

1.5 Incident Response Generator

Purpose: Document actions for compliance and auditing.

Implementation Details (e.g.):

- **Tools**:
    - **Elasticsearch**: Store logs and generate dashboards.
    - **Jinja2**: Auto-generate reports.
    - **Etc**
- **Example Report Template**:
```
## Incident Report
**Time**: 2023-10-05 14:30 UTC
**Vulnerability**: CVE-2014-0160 (OpenSSL Heartbleed)
**Action Taken**: Upgraded OpenSSL to 1.1.1.
**Test Results**: All unit/integration tests passed.
**Residual Risk**: None.
```

---

2. System Integration and Component Interaction

1. **Detection**:
    - **Risk Monitor** detects outdated OpenSSL → triggers **Patch Suggester**.
2. **Patch Generation**:
    - **Patch Suggester** drafts upgrade command → sends to **Validation Sandbox**.
3. **Testing**:
    - **Sandbox** clones production, runs tests → approves/rejects patch.
4. **Deployment**:
    - **Deployment Engine** applies patch via rolling update.
5. **Reporting**:
    - **Incident Generator** logs action → notifies teams via Slack.

---

3. Evaluation Criteria

1. **Time-to-Patch**: Average time from detection to deployment (target: <1 hour).
2. **Patch Success Rate**: Percentage of patches applied without rollback.
3. **False Positives**: Rate of unnecessary patches suggested by LLM.
4. **Compliance**: Audit-ready reports generated for 100% of actions.

---

4. Ethical and Operational Considerations

- **Human Oversight**: Require approval for critical systems (e.g., healthcare, etc).
- **Bias Mitigation**: Audit LLM suggestions against industry standards (e.g., CIS Benchmarks, etc).
- **Transparency**: Log all automated actions for traceability.

---

5. Tools and Resources (e.g.)

- **Monitoring**: Snyk, ScoutSuite, NVD API, etc.
- **LLM**: OpenAI GPT-4, Claude, CodeLlama, etc.
- **Deployment**: Kubernetes, Terraform, GitHub Actions, etc.
- **Reporting**: Elasticsearch, Jinja2, etc.
