# GET

Whenever we visit any URL, our browsers default to a GET request to obtain the remote resources hosted at that URL. Once the browser receives the initial page it is requesting; it may send other requests using various HTTP methods. This can be observed through the Network tab in the browser devtools, as seen in the previous section.

> **Exercise:** Pick any website of your choosing, and monitor the Network tab in the browser devtools as you visit it to understand what the page is performing. This technique can be used to thoroughly understand how a web application interacts with its backend, which can be an essential exercise for any web application assessment or bug bounty exercise.

HTTP Basic Auth

When we visit the exercise found at the end of this section, it prompts us to enter a username and a password. Unlike the usual login forms, which utilize HTTP parameters to validate the user credentials (e.g. POST request), this type of authentication utilizes a `basic HTTP authentication`, which is handled directly by the webserver to protect a specific page/directory, without directly interacting with the web application.

To access the page, we have to enter a valid pair of credentials, which are `admin:admin` in this case:
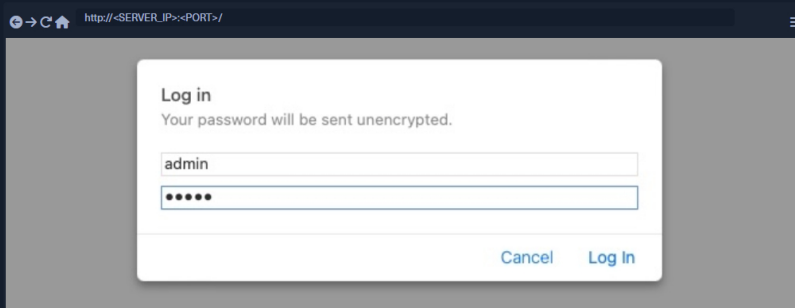
Once we enter the credentials, we would get access to the page:
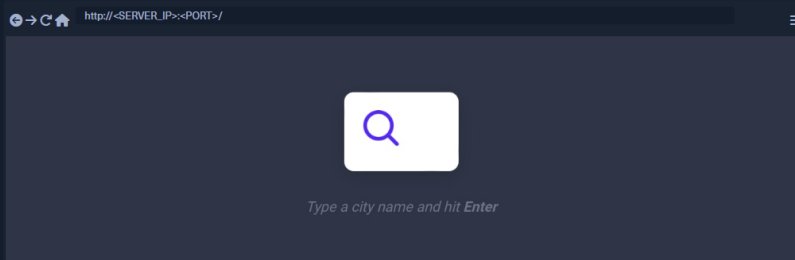
Let's try to access the page with cURL, and we'll add `-i` to view the response headers:

```
                                    GET
MisaelMacias@htb[/htb]$ curl -i http://<SERVER_IP>:<PORT>/
HTTP/1.1 401 Authorization Required
Date: Mon, 21 Feb 2022 13:11:46 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: no-cache, must-revalidate, max-age=0
WWW-Authenticate: Basic realm="Access denied"
Content-Length: 13
Content-Type: text/html; charset=UTF-8

Access denied
```

As we can see, we get `Access denied` in the response body, and we also get `Basic realm="Access denied"` in the `WWW-Authenticate` header, which confirms that this page indeed uses `basic HTTP auth`, as discussed in the Headers section. To provide the credentials through cURL, we can use the `-u` flag, as follows:

```
                                    GET
MisaelMacias@htb[/htb]$ curl -u admin:admin http://<SERVER_IP>:<PORT>/

<!DOCTYPE html>
<html lang="en">

<head>
...SNIP...
```

This time we do get the page in the response. There is another method we can provide the `basic HTTP auth` credentials, which is directly through the URL as (`username:password@URL`), as we discussed in the first section. If we try the same with cURL or our browser, we do get access to the page as well:

```
                                    GET
```

My Workstation

OFFLINE

▶ Start Instance

∞ / 1 spawns left

```
MisaelMacias@htb[/htb]$ curl http://admin:admin@<SERVER_IP>:<PORT>/

<!DOCTYPE html>
<html lang="en">

<head>
...SNIP...
```

We may also try visiting the same URL on a browser, and we should get authenticated as well.

> **Exercise:** Try to view the response headers by adding -i to the above request, and see how an authenticated response differs from an unauthenticated one.

HTTP Authorization Header

If we add the `-v` flag to either of our earlier cURL commands:

```
                                          GET

MisaelMacias@htb[/htb]$ curl -v http://admin:admin@<SERVER_IP>:<PORT>/

*   Trying <SERVER_IP>:<PORT>...
* Connected to <SERVER_IP> (<SERVER_IP>) port PORT (#0)
* Server auth using Basic with user 'admin'
> GET / HTTP/1.1
> Host: <SERVER_IP>
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.77.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 21 Feb 2022 13:19:57 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Cache-Control: no-store, no-cache, must-revalidate
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Length: 1453
< Content-Type: text/html; charset=UTF-8
<

<!DOCTYPE html>
<html lang="en">

<head>
...SNIP...
```

As we are using `basic HTTP auth`, we see that our HTTP request sets the `Authorization` header to `Basic YWRtaW46YWRtaW4=`, which is the base64 encoded value of `admin:admin`. If we were using a modern method of authentication (e.g. `JWT`), the `Authorization` would be of type `Bearer` and would contain a longer encrypted token.

Let's try to manually set the `Authorization`, without supplying the credentials, to see if it does allow us access to the page. We can set the header with the `-H` flag, and will use the same value from the above HTTP request. We can add the `-H` flag multiple times to specify multiple headers:

```
                                          GET

MisaelMacias@htb[/htb]$ curl -H 'Authorization: Basic YWRtaW46YWRtaW4=' http://<SERVER_IP>:<PORT>/

<!DOCTYPE html
<html lang="en">

<head>
...SNIP...
```

As we see, this also gave us access to the page. These are a few methods we can use to authenticate to the page. Most modern web applications use login forms built with the back-end scripting language (e.g. PHP), which utilize HTTP POST requests to authenticate the users and then return a cookie to maintain their authentication.
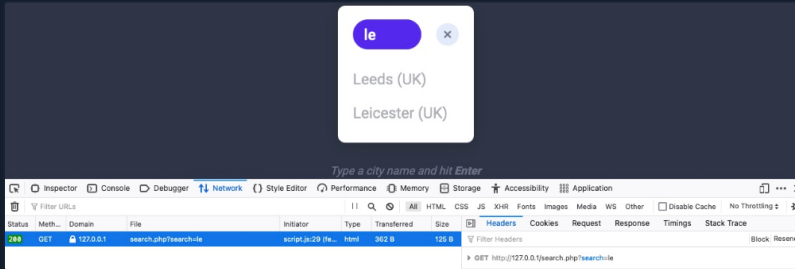
GET Parameters

Once we are authenticated, we get access to a `City Search` function, in which we can enter a search term and get a list of matching cities:
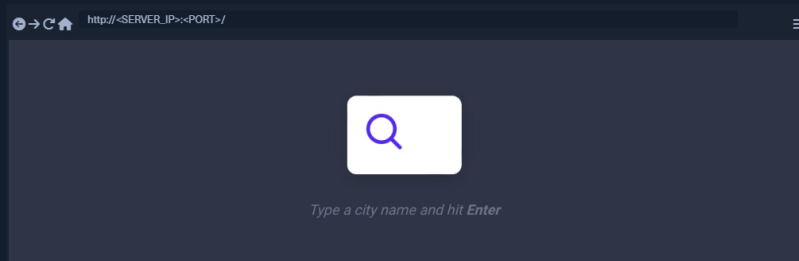
As the page returns our results, it may be contacting a remote resource to obtain the information, and then display them on the page. To verify this, we can open the browser devtools and go to the Network tab, or use the shortcut [CTRL+SHIFT+E] to get to the same tab. Before we enter our search term and view the requests, we may need to click on the `trash` icon on the top left, to ensure we clear any previous requests and only monitor newer requests:
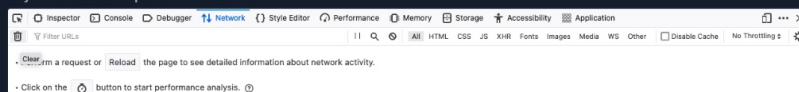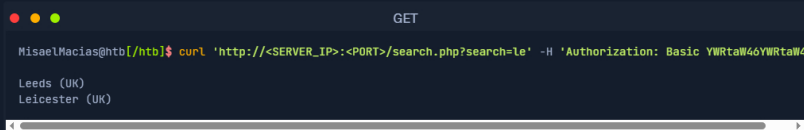
After that, we can enter any search term and hit enter, and we will immediately notice a new request being sent to the backend:

When we click on the request, it gets sent to `search.php` with the GET parameter `search=le` used in the URL. This helps us understand that the search function requests another page for the results.

Now, we can send the same request directly to `search.php` to get the full search results, though it will probably return them in a specific format (e.g. JSON) without having the HTML layout shown in the above screenshot.

To send a GET request with cURL, we can use the exact same URL seen in the above screenshots since GET requests place their parameters in the URL. However, browser devtools provide a more convenient method of obtaining the cURL command. We can right-click on the request and select `Copy>Copy as cURL`. Then, we can paste the copied command in our terminal and execute it, and we should get the exact same response:

```
● ● ●                                    GET

MisaelMacias@htb[/htb]$ curl 'http://<SERVER_IP>:<PORT>/search.php?search=le' -H 'Authorization: Basic YWRtaW46YWRtaW4

Leeds (UK)
Leicester (UK)
```
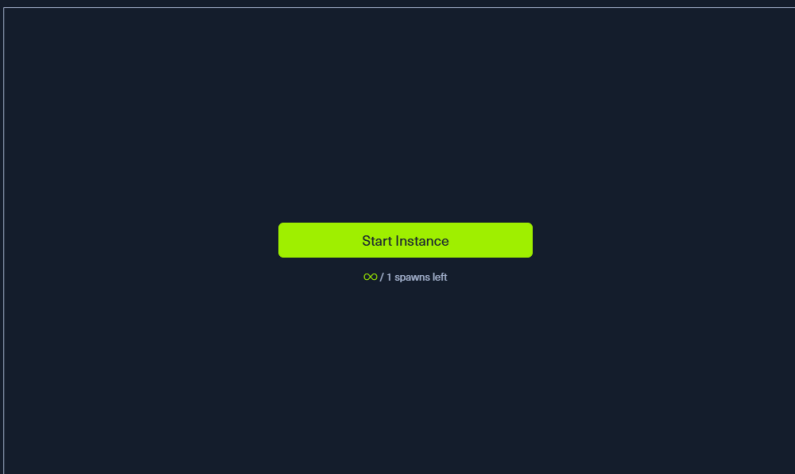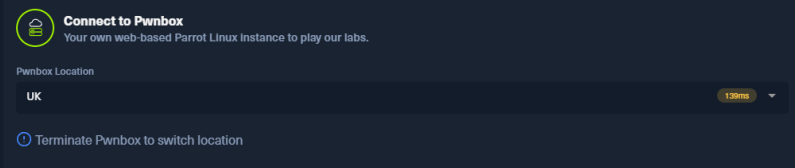
> **Note:** The copied command will contain all headers used in the HTTP request. However, we can remove most of them and only keep necessary authentication headers, like the `Authorization` header.

We can also repeat the exact request right within the browser devtools, by selecting `Copy>Copy as Fetch`. This will copy the same HTTP request using the JavaScript Fetch library. Then, we can go to the JavaScript console tab by clicking [`CTRL+SHIFT+K`], paste our Fetch command and hit enter to send the request:

As we see, the browser sent our request, and we can see the response returned after it. We can click on the response to view its details, expand various details, and read them.

Connect to Pwnbox
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

| UK | 139ms ▼ |

ⓘ Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Questions

Answer the question(s) below to complete this Section and earn cubes!

Target(s): Click here to spawn the target system!

Authenticate to with user "admin" and password "admin"

+ 2 ⬡ The exercise above seems to be broken, as it returns incorrect results. Use the browser devtools to see what is the request it is sending when we search, and use cURL to search for 'flag' and obtain the flag.

HTB{curl_g3773r}

Submit   Hint

← Previous   Next →

✓ Mark Complete & Next