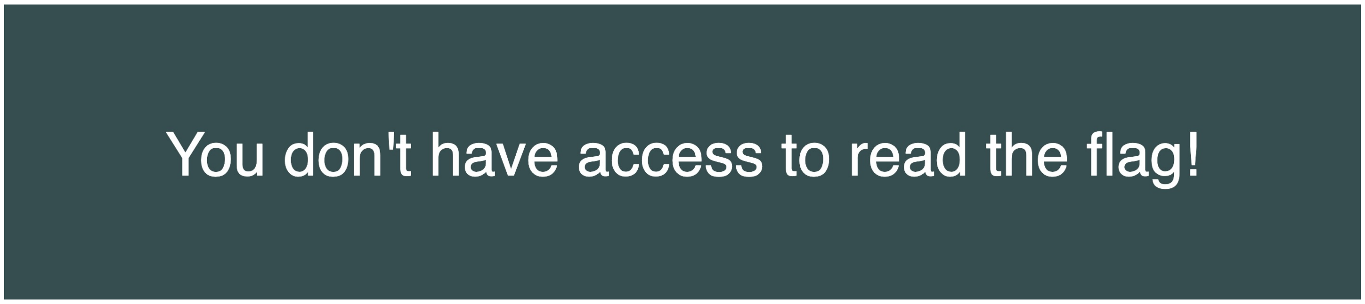
Parameter Fuzzing

Parameter Fuzzing - GET

If we run a recursive `ffuf` scan on `admin.academy.htb`, we should find `http://admin.academy.htb:PORT/admin/admin.php`. If we try accessing this page, we see the following:



You don't have access to read the flag!

That indicates that there must be something that identifies users to verify whether they have access to read the `flag`. We did not login, nor do we have any cookie that can be verified at the backend. So, perhaps there is a key that we can pass to the page to read the `flag`. Such keys would usually be passed as a `parameter`, using either a `GET` or a `POST` HTTP request. This section will discuss how to fuzz for such parameters until we identify a parameter that can be accepted by the page.

Tip: Fuzzing parameters may expose unpublished parameters that are publicly accessible. Such parameters tend to be less tested and less secured, so it is important to test such parameters for the web vulnerabilities we discuss in other modules.

GET Request Fuzzing

Similarly to how we have been fuzzing various parts of a website, we will use `ffuf` to enumerate parameters. Let us first start with fuzzing for `GET` requests, which are usually passed right after the URL, with a `?` symbol, like:

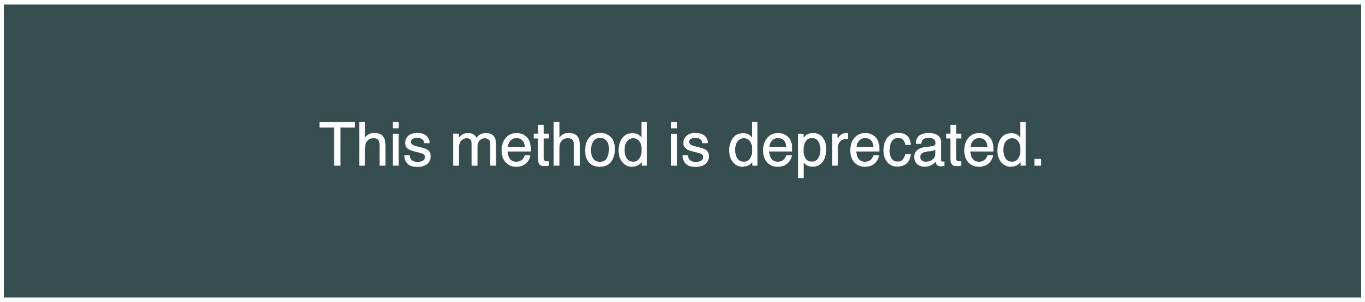
- `http://admin.academy.htb:PORT/admin/admin.php?param1=key`.

So, all we have to do is replace `param1` in the example above with `FUZZ` and rerun our scan. Before we can start, however, we must pick an appropriate wordlist. Once again, `SecLists` has just that in `/opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt`. With that, we can run our scan.

Once again, we will get many results back, so we will filter out the default response size we are getting.

```
mayala@htb[/htb] $ ffuf -w /opt/useful/SecLists/Discovery/Web-Content/burp-
parameter-names.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php?
FUZZ=key -fs xxx /'___\ /'___\ /'___\ /\ \__/ /\ \__/ __ __ /\ \__/ \ \ ,__\\ \
,__\/\ \/\ \ \ \ ,__\ \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/ \ \_\ \ \_\ \ \____/ \
\_\ \/_/ \/_/ \/___/ \/_/ v1.1.0-git
________________________________________________ :: Method : GET :: URL :
http://admin.academy.htb:PORT/admin/admin.php?FUZZ=key :: Wordlist : FUZZ:
/opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt :: Follow
redirects : false :: Calibration : false :: Timeout : 10 :: Threads : 40 ::
Matcher : Response status: 200,204,301,302,307,401,403 :: Filter : Response
size: xxx ________________________________________________ <...SNIP...> [Status:
xxx, Size: xxx, Words: xxx, Lines: xxx]
```

We do get a hit back. Let us try to visit the page and add this `GET` parameter, and see whether we can read the flag now:



This method is deprecated.

As we can see, the only hit we got back has been `deprecated` and appears to be no longer in use.

Parameter Fuzzing - POST

The main difference between `POST` requests and `GET` requests is that `POST` requests are not passed with the URL and cannot simply be appended after a `?` symbol. `POST` requests are passed in the `data` field within the HTTP request. Check out the Web Requests module to learn more about HTTP requests.

To fuzz the `data` field with `ffuf`, we can use the `-d` flag, as we saw previously in the output of `ffuf -h`. We also have to add `-X POST` to send `POST` requests.

Tip: In PHP, "POST" data "content-type" can only accept "application/x-www-form-urlencoded". So, we can set that in "ffuf" with "-H 'Content-Type: application/x-www-form-urlencoded'".

So, let us repeat what we did earlier, but place our `FUZZ` keyword after the `-d` flag:

```
mayala@htb[/htb] $ ffuf -w /opt/useful/SecLists/Discovery/Web-Content/burp-
parameter-names.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php -X
POST -d 'FUZZ=key' -H 'Content-Type: application/x-www-form-urlencoded' -fs xxx
/'___\ /'___\ /'___\ /\ \__/ /\ \__/ __ __ /\ \__/ \ \ ,__\\ \ ,__\/\ \/\ \ \ \
,__\ \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/ \ \_\ \ \_\ \ \____/ \ \_\ \/_/ \/_/
\/___/ \/_/ v1.1.0-git ________________________________________________ ::
Method : POST :: URL : http://admin.academy.htb:PORT/admin/admin.php :: Wordlist
: FUZZ: /opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt ::
Header : Content-Type: application/x-www-form-urlencoded :: Data : FUZZ=key ::
Follow redirects : false :: Calibration : false :: Timeout : 10 :: Threads : 40
:: Matcher : Response status: 200,204,301,302,307,401,403 :: Filter : Response
size: xxx ________________________________________________ id [Status: xxx,
Size: xxx, Words: xxx, Lines: xxx] <...SNIP...>
```

As we can see this time, we got a couple of hits, the same one we got when fuzzing `GET` and another parameter, which is `id`. Let's see what we get if we send a `POST` request with the `id` parameter. We can do that with `curl`, as follows:

```
mayala@htb[/htb] $ curl http://admin.academy.htb:PORT/admin/admin.php -X POST -d
'id=key' -H 'Content-Type: application/x-www-form-urlencoded' <div
class='center'><p>Invalid id!</p></div> <...SNIP...>
```

As we can see, the message now says `Invalid id!`.

Value Fuzzing

After fuzzing a working parameter, we now have to fuzz the correct value that would return the `flag` content we need. This section will discuss fuzzing for parameter values, which should be fairly similar to fuzzing for parameters, once we develop our wordlist.

Custom Wordlist

When it comes to fuzzing parameter values, we may not always find a pre-made wordlist that would work for us, as each parameter would expect a certain type of value.

For some parameters, like usernames, we can find a pre-made wordlist for potential usernames, or we may create our own based on users that may potentially be using the website. For such cases, we can look for various wordlists under the `seclists` directory and try to find one that may contain values matching the parameter we are targeting. In other cases, like custom parameters, we may have to develop our own wordlist. In this case, we can guess that the `id` parameter can accept a number input of some sort. These ids can be in a custom format, or can be sequential, like from 1-1000 or 1-1000000, and so on. We'll start with a wordlist containing all numbers from 1-1000.

There are many ways to create this wordlist, from manually typing the IDs in a file, or scripting it using Bash or Python. The simplest way is to use the following command in Bash that writes all numbers from 1-1000 to a file:

```
mayala@htb[/htb]$ for i in $(seq 1 1000); do echo $i >> ids.txt; done
```

Once we run our command, we should have our wordlist ready:

```
mayala@htb[/htb]$ cat ids.txt 1 2 3 4 5 6 <...SNIP...>
```

Now we can move on to fuzzing for values.

Value Fuzzing

Our command should be fairly similar to the `POST` command we used to fuzz for parameters, but our `FUZZ` keyword should be put where the parameter value would be, and we will use the `ids.txt` wordlist we just created, as follows:

```
mayala@htb[/htb]$ ffuf -w ids.txt:FUZZ -u http://admin.academy.htb:PORT/admin/admin.php -X POST -d 'id=FUZZ' -H 'Content-Type: application/x-www-form-urlencoded' -fs xxx /'___\ /'___\ /'___\ /\ \__/ /\ \__/ __ __ /\ \__/ \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\ \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/ \ \_\ \ \_\ \ \____/ \ \_\ \/_/ \/_/ \/___/ \/_/ v1.0.2 ________________________________________________ :: Method : POST :: URL : http://admin.academy.htb:30794/admin/admin.php :: Header : Content-Type: application/x-www-form-urlencoded :: Data : id=FUZZ :: Follow redirects : false :: Calibration : false :: Timeout : 10 :: Threads : 40 :: Matcher : Response status: 200,204,301,302,307,401,403 :: Filter : Response size: xxx ________________________________________________ <...SNIP...> [Status: xxx, Size: xxx, Words: xxx, Lines: xxx]
```

We see that we get a hit right away. We can finally send another `POST` request using `curl`, as we did in the previous section, use the `id` value we just found, and collect the flag.