CyberSecurity assignment 4

1. Structured query language injection (SQLI): also known as SQL injection, is a type of injection attack in which, The attacker sends malicious data that can trick the SQL interpreter into executing malicious SQL statements; this allows the attacker to add, delete, read, and modify data in the database, bypass authentication and authorization, and access admin privileges.

   patterns of queries that indicate an SQL Injection attack:

   - **Line comment:**
     SQL interpreter stops executing rest of the statement after line comment. It is represented by '--'.

     Sample Attack:
     User input:
     req.username = admin"--
     req.password = 1234567890
     SQL query:
     "SELECT * FROM users WHERE username = "+ req.username + "password = " req.password + ";"
     Explanation:
     feed the above user input into the query and feed the resultant query into the database. It will return all accounts with username admin without checking the password due to the comments. This allows the user to log in to the application as an admin.

   - **Stacking Query:**
     executing multiple statements in the same call to the database server, can be used to inject malicious SQL statements by using the statement terminator character ';'.

     Sample attack:
     User input:
     req.id = 1; DROP users--
     SQL query:
     "SELECT * FROM products WHERE id = " + req.id + ";"
     Explanation:
     feed the above user input into the query and feed the resultant query into the database. It will return the product record with id one and drops the users table.

   - **Getting more information than intended:**

Sample attack:
User input:
req.id = 999 or 1=1
SQL query:
“SELECT * FROM employees WHERE employeeid = ” + req.id + “;”
Explaination:
Here an attacker is able to get all employee’s data, because in the attackers input there is – or 1=1 – which is true always.

- **Union injection:**
Sample attack:
User input:
req.id = 999 UNION SELECT username, password FROM users
SQL query:
“SELECT itemname, itemdescription FROM items WHERE itemid = ” + req.id + “;”
Explaination:
when above query Is fed into the database interpreter, it returns with a table which contains all records from users table and record from items table whose itemid is 999.

- **Getting information regarding database tables, table’s coloumns:**
We can use specific SQL queries to get information regarding tables present in the database and coloumns present in the tables and we can also get information regarding version number of database server.

  Sample attack:
  Query1:
  SELECT name FROM sysobjects WHERE xtype = ‘U’
  Query2:
  SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = ‘tabelnamecolumnnames);
  Explaination:
  we can use Query1 to get user defined tables from the database. We can use Query2 to get coloumn names.

- **If Statements:**
Get response based on an if statement. This is one of the key points of Blind SQL Injection, also can be very useful to test simple stuff blindly and accurately.

  Sample attack:
  User input:
  req.username = hello" UNION (SELECT IF("Alfreds Futterkiste" in (select username from users),"yes","no"));

SQL query:

"SELECT username FROM users WHERE username = " + req.username + ";"

Explaination:

here we need to know more information regarding tables and columns. Which we can get by using blind injection attacks.

2. Demonstrate the effectiveness of same-origin policy. What attacks does it prevent:

Origin in same origin policy is defined by protocol, domain, port of a URL.

Eg: https://www.youtube.com:443/

Here protocol is https.

Domain is www.youtube.com.

Port is 443.

Effectiveness of same origin policy:

Let web content received by accesing a malicious web site be:

Let web address be: http://www.hfacebook.com:80/

```
<html>
<head>…</head>
<script>
submit = document.getElementById("frame").getElementById("submit");
submit.onclick = function(){
// access credentials from here.
}
</script>
<body>
<iframe id="frame"  src="https://www.facebook.com:443/"></iframe>
</body>
</html>
```

Here same origin policy doesnot allow to access dom of the iframe.

Attacks prevented by Same origin policy:

It prevent, malicious websites from reading confidential data from other websites. But it also prevent from reading information affered by genuine website.

3. What are the top ten latest OWASP API application security attacks in API level. Explain each with example.

1. API:1 :- Broken object level authorization:
   Here applications will have authentication but doesnot have object level authorization. A authenticated person can access both his and others information by changing url.
   Eg: GET /empinfo/{id}
   Here instead of your id you can change it to others id and access their information. Avoid it by implementing access control where each user can only access their information.

2. API:2 :- Broken Authentication: Unable to distinguish between actual user and hacker.
   For example, lets assume that an hacker stole the session key of the genuine user and used it to access the server. Here server has allowed hacker to access the resources because it is not able to distinguish between them. Avoid it by using ip address in the generation of session key so that when an hacker uses the session we can detect the change in ip address from where server is accesed.

3. API:3 :- Excessive data exposure: exposing more than required data by an api. So that it can be used by an attacker.
   Eg: GET /empinfo?name={name}
   When attacker search for an employee by name=victimemployee. api returns {name: victimemloyee,password: 1234567890,dep:"CSE"}
   Avoid it by having a gateway which make sures that it does not allow for excess information to pass.

4. API:4 :- Lack of Resources and Rate Limiting: not limiting the file size in the request or response can create the shortage of resources on server which in turn becomes a denial of service for other users.
   Eg:

Size of a file that has been requested to store on server is big, which has overwhelmed the resources on the server, which in turn lead to the denial of service by server for other users. Avoid it by having a gateway which ensures the limit on file size.

5. API:5 :- Broken Function level authorization: some functionalities of the server are meant to be accessed by only authorized users but due to broken authorization every one can access the functionalities by using the api.
   Can be avoided by having a gateway which implements the access control.

6. API:6 :- Mass Assignment: using put operation to update record can lead to the problems.
   Eg: {name:”test”,balance:”99”}
   Update it by
   {name:”test”,balance:”999”}
   Avoid it by using APi gateway which does not allow such assignments from happening.

7. API:7 :- Security Missconfiguration: displaying in depth error messages, using http instead of https, poor CORS policy enforcement.
   Avoid it by having strict CORS policy enforcement, and using a gateway which doesnot allow to display indept information to the user.

8. API:8 :- Injection: directly using user input to access databases (any sql or nosql) can lead expose more information than required and can lead to bypass authentication and authorization.
   Avoid it by using whitelisted pattern and sanitizing input.

9. API:9 :- Improper assets management: occurs when different versions of api is managed and an intruder uses older version to get more information.
   Avoid it by using api gateway which redirects user to the new version of api.

10. API:10 :- Insufficiant logging and monitoring: inssuficient logging can lead to not identifiy intruder.
    Eg: when there are resources which can only accessed by authorized users and there is no loging of information regarding who accessed the it cannot be identified and rectified.

4. Explain how broken authentication leads to attacks:

Broken Authentication: it is caused by poor implemenatation of authentication and session management.

These vulnerabilities are exploited by attackers to do

1. Credential stuffing.
2. Session hijacking.
3. Password spraing.

Etc.

When attackers access a database filled with unencrypted emails and passwords, they frequently sell or give away the list for other attackers to use. These attackers then use botnets for brute-force attacks that test credentials stolen from one site on different accounts. This tactic often works because people frequently use the same password across applications.

Without appropriate safeguards, web applications are vulnerable to session hijacking, in which attackers use stolen session IDs to impersonate users' identities.

Password spraying is a little like credential stuffing, but instead of working off a database of stolen passwords, it uses a set of weak or common passwords to break into a user's account.

5. **Look at the history of AWS attacks on amazon cloud. Take any five of them. Analyze them from perspective of what was the attack? What vulnerability led to the same. What can be done to mitigate the occurrence of same again:**

Any Five attacks on AWS:

1.
Attack: Phishing.
Vulnerability: the weakest link in the system.
Mitigation: anti-spyware and firewall settings should be used to prevent phishing attacks and users should update the program regularly.

2.
Attack: SQL injection.
Vulnerability: was in sql query interpreter.
Mitigation: sanitize all inputs and if possible use only whielisted patterns as inputs.

3.
Attack: Distributed Denial of Service (DDOS)
Vulnerability: accepting multiple requests from a distributed set of sytems which depletes the resources of the server.
Mitigation: Amazon shield - AWS Shield Standard uses various techniques like priority-based traffic shaping which are automatically engaged when a well-defined DDoS attack signature is detected

4.

Attack: Malware.
Vulnerability: Some systems allow code executed by a user to access all rights of that user, which is known as over-privileged code This makes users vulnerable to malware.
Mitigation:
Keeping the computer and software updated
Think twice before clicking links or downloading anything
Don't trust pop-up windows that ask you to download software
Limit your file-sharing
Use antivirus software

5.
(happened on cloud but don’t know whether it happened on AWS or not)
Attack:Improper asset Management
Vulnerability: Maintaining or using older versions of software
Mitigation :By blocking the calls if the request comes from the older versions of software

6. **Create a sample CRUD based REST API using Flask or Django or Java or any framework you are comfortable with. Briefly discuss potential attacks on same. Hands on - This is to be demonstrated with code . Screenshots and working code should be submitted:**

sample crud based RESTapi using flask and sql database:
it’s a token management system and more information is in below link.
api details are in README.md file of the below link.
potential api attacks are:

1. Injection: injection.PNG
2. Broken Authentication: broken_authentication.PNG
3. Insufficient logging: insufficient_logging.PNG
4. Security misconfiguration: security_misconfiguration.PNG
5. Lack of resources or rate limiting: lackofresourcesorratelimiting.PNG

Link to the code and screen shots: https://github.com/Dileepredd/cybersecurity-assignment-4