Tech Stack

- MySQl
- Springboot
- React.js

MySQL Vulnerabilities

1. SQL Injection: It is among the most common and perilous attacks, in this type of attack the attackers steal information or foist data loss by attacking the database. Basically, it is an injection-type attack where the attacker runs malicious SQL queries that could have various serious implications such as losing data or even data stealing.

```
SELECT * FROM mutable WHERE username = “UserName001”
AND password = ‘*’ OR ‘1’ = ’1’.
```

2. Improper Input Validation: Improper Input Validation is a dangerous type of attack where a malicious user carries out an attack on web servers or their instances such as MySQL. These attacks are used to make the instances, services, or network resources inaccessible momentarily or even permanently by disrupting the host, system, or the instance it is connected to. In MySQL, it can make the instance of MySQL crash hence making it momentarily inaccessible by any of the services that are using it as its data source.

The other issue with this type of vulnerability is that it allows remote authenticated users to cause a DoS attack by using a crafted SELECT statement along with a UpdateXML() command with many unique nested elements. The result of this is a more susceptible MySQL to DoS vulnerability. An attacker could exploit this flaw to takedown the whole database and its instances, rendering other services useless and making them inaccessible to the user.

```
$mysql->query("SELECT UpdateXML('<a>$a<b>ccc</b>
<d></d></a>', '/a', '<e>fff</e>') AS val1");
```

3. Concurrent Execution using Shared Resources with Improper Synchronisation or Race Condition: Concurrent Execution using Shared Resources with Improper Synchronisation is an undesired condition that happens when a system tries to run two or more than two operations concurrently, though in a majority of the system the operations are performed in the right order or timing because of the uncontrolled events to make sure that the operations are done effectively.
In MySQL, this can give rise to a race condition, which can be considered a serious problem. It allows a local user to access the database and after that, they can use privilege escalation or escalate their user privileges and after changing their user privileges, they can carry out an arbitrary code execution attack as a local user of the database.

4. Permission, Privileges, and Access Controls: It is an old vulnerability that now has been patched. This vulnerability allowed the malicious users to override the config file of MySQL with numerous settings, so these settings could be implemented when MySQL is started next time.

5. DNS Injection in ghost: This vulnerability was present in ghost (a triggerless schema migration tool for MySQL) versions before version 1.1.3 which had a file path traversal or directory traversal vulnerabilities. In order to exploit it, the attacker either needs to have access to the target host or they need to trick the admin to run a crafted ghost command on the host running ghost along with the network access from the target host running ghost to the attack's malicious

MySQL server. The -database parameter must also be properly sanitized to avoid this.

```
./gh-ost -user test -password -test -alter
 test -table test -database “test?allowAllFiles=true&”
```

Springboot Vulnerabilities

CVE-2023-34054: Reactor Netty HTTP Server Metrics DoS Vulnerability

In Reactor Netty HTTP Server, versions 1.1.x prior to 1.1.13 and versions 1.0.x prior to 1.0.39, it is possible for a user to provide specially crafted HTTP requests that may cause a denial-of-service (DoS) condition.

Specifically, an application is vulnerable if Reactor Netty HTTP Server built-in integration with Micrometer is enabled.

CVE-2023-34055: Spring Boot server Web Observations DoS Vulnerability:

In Spring Boot versions 2.7.0 - 2.7.17, 3.0.0-3.0.12 and 3.1.0-3.1.5, it is possible for a user to provide specially crafted HTTP requests that may cause a denial-of-service (DoS) condition.

Specifically, an application is vulnerable when all of the following are true:

- the application uses Spring MVC or Spring WebFlux
- `org.springframework.boot:spring-boot-actuator` is on the classpath

CVE-2023-34035: Authorization rules can be misconfigured when using multiple servlets

Spring Security versions **5.8** prior to **5.8.5**, **6.0** prior to **6.0.5** and **6.1** prior to **6.1.2** could be susceptible to authorization rule misconfiguration if the application uses `requestMatchers(String)` or `requestMatchers(HttpMethod, String)` and multiple servlets, one of them being Spring MVC's DispatcherServlet.

(`DispatcherServlet` is a Spring MVC component that maps HTTP endpoints to methods on `@Controller`-annotated classes.)

Specifically, an application is vulnerable when all of the following are true:

- Spring MVC is on the classpath
- Spring Security is securing more than one servlet in a single application (one of them being Spring MVC's `DispatcherServlet`)
- The application uses `requestMatchers(String)` or `requestMatchers(HttpMethod, String)`

An application is not vulnerable if any of the following is true:

- The application does not have Spring MVC on the classpath
- The application secures no servlets other than Spring MVC's `DispatcherServlet`
- The application does not use `requestMatchers(String)` or `requestMatchers(HttpMethod, String`

CVE-2023-20862: Empty SecurityContext Is Not Properly Saved Upon Logout

In Spring Security, versions 5.7.x prior to 5.7.8, versions 5.8.x prior to 5.8.3, and versions 6.0.x prior to 6.0.3, the logout support does not properly clean the security context if using serialized versions. Additionally, it is not possible to explicitly save an empty security context to the HttpSessionSecurityContextRepository. This vulnerability can keep users authenticated even after they performed logout.

Specifically, an application is vulnerable when any of the following is true:

- You are using the SecurityContextHolderFilter or requireExplicitSave(true) and you are using Spring Security's logout support with serialized sessions (e.g. Spring Session) and invalidateHttpSession(false)
- You are logging users out manually by saving an empty SecurityContext into the HttpSessionSecurityContextRepository
- You have a custom SecurityContextRepository that does not rely on the HttpSession

An application is not vulnerable if any of the following is true:

- You are still using the deprecated SecurityContextPersistenceFilter or requireExplicitSave(false)
- You are using Spring Security's logout support with in-memory sessions.
- You are not saving an empty SecurityContext into the HttpSessionSecurityContextRepository

CVE-2023-20863: Spring Expression DoS Vulnerability

In Spring Framework versions 6.0.0 - 6.0.7, 5.3.0 - 5.3.26, 5.2.0.RELEASE - 5.2.23.RELEASE, and older unsupported versions, it is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service (DoS) condition.

CVE-2023-20866: Session ID can be logged to the standard output stream in Spring Session

In Spring Session version 3.0.0, the session id can be logged to the standard output stream. This vulnerability exposes sensitive information to those who have access to the application logs and can be used for session hijacking.

React.js Vulnarabilities

SQL Injection:

SQL Injection (SQLi) is a widely known web application attack. The cybercriminal intends to perform database manipulation logically to access sensitive information that is not supposed to be displayed. Attackers try to sneak into that sensitive information to collect phone numbers, payment details, addresses, passwords, and other credentials. This technique allows the attackers to manage access to the server, pull the data, and manipulate the values in the database. Apart from data modification, hackers can also delete the data. Let us take an example. Let us take an example where the code will search for the current user-ID and the login matching the employee name, where the owner is the current user-ID.

Cross-Site Scripting (XSS) Vulnerabilities:

The comprehensive rendering feature of React makes it a preferable choice over other JavaScript libraries and frameworks. But this rendering feature also drags React-based apps to the most widely exploited vulnerability, cross-site scripting (XSS). Cross-site scripting leverages malicious client-side scripts by injecting them into web applications. When the users trigger those scripts, the attackers gain control over the app and steal sensitive information from the web application.

Injecting malicious scripts into the react app will make the app release some internal app data. Therefore, React developers should prevent the application from running the script.

Broken Authentication

Broken authentication is a weakness through which attackers can capture one or multiple accounts when authentication or session management is poorly implemented in progressive web apps. This vulnerability helps the attacker take over multiple user accounts, letting the attacker possess the same privileges and access control as the target user.

Attackers usually exploit such a React security vulnerability by detecting the authentication solution or bypassing them. The security team labels the authentication as broken when the cybercriminal can compromise users' passwords, session tokens, digital identities, or account information from the React app.

Cross-site Request Forgery (CSRF or XSRF)

CSRF is another React web application vulnerability allowing attackers to persuade users to perform unintentional actions without their direct consent. It does not steal the identity of the legitimate user but acts against their will. For rendering such attacks, the attacker sends an email or a web link convincing the victim to achieve a state-changing request in the application.

Zip Slip

It is another popular React app vulnerability where the malicious actor exploits the app by submitting zip files having malicious or arbitrary code within them. React developers enable adding zip files to decrease the file size while they get uploaded. When the app unzips the archive, its malicious file(s) can overwrite other files or perform arbitrary code execution. Attackers can either harm the files existing in the target system or gain remote access to the system.

Here is a Zip slip code example demonstrating a ZipEntry path merges to a destination directory without validating that path. Researchers and security professionals have found similar codes in different repositories across many apps.

XML External Entities (XXE)

Attackers can perform XML External Entities attacks on React web applications that parse XML input. Here the outdated XML parsers of your React app become vulnerable. Such vulnerability can lead an attacker to perform denial of service, port scanning, server-side request forgery, etc. Such attacks occur when an XML input gets referred to an external entity but has a weakly configured XML parser. Here are some examples where attackers are attempting XEE on React web applications that parse XML input.