

**Finding Name:** Potential SQL injection.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Team** | **Role** | **Project** | **Quality Assurance** | **Is this a re-tested Finding?** |
| Gaurish Bhatia | SCR | Junior team member | Ontrack | Jaspriya Kaur and Payas Paul |  |

|  |
| --- |
| **Was this Finding Successful?** |
| Yes |

**Finding Description**

**An SQL injection is a type of attack in which the attacker gains unauthorized access to the database by use of SQL queries which could be directly injected in the given vulnerable code which is not sanitized enough in order to bypass any of the code, which is injected through the input. Specifically, in this case, the ‘subquery’ used in the ‘joins’ method, which could lead to a potential injection.**

**Risk Rating**  
Impact: Major  
Likelihood: High

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Impact values** | | | | |
| **Very Minor** | **Minor** | **Significant** | **Major** | **Severe** |
| Risk that holds little to no impact. Will not cause damage and regular activity can continue. | Risk that holds minor form of impact, but not significant enough to be of threat. Can cause some damage but not enough to impede regular activity. | Risk that holds enough impact to be somewhat of a threat. Will cause damage that can impede regular activity but will be able to run normally. | Risk that holds major impact to be of threat. Will cause damage that will impede regular activity and will not be able to run normally. | Risk that holds severe impact and is a threat. Will cause critical damage that can cease activity to be run. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Likelihood** | | | | |
| **Rare** | **Unlikely** | **Moderate** | **High** | **Certain** |
| Event may occur and/or if it did, it happens in specific circumstances. | Event could occur occasionally and/or could happen (at some point) | Event may occur and/or happens. | Event occurs at times and/or probably happens a lot. | Event is occurring now and/or happens frequently. |

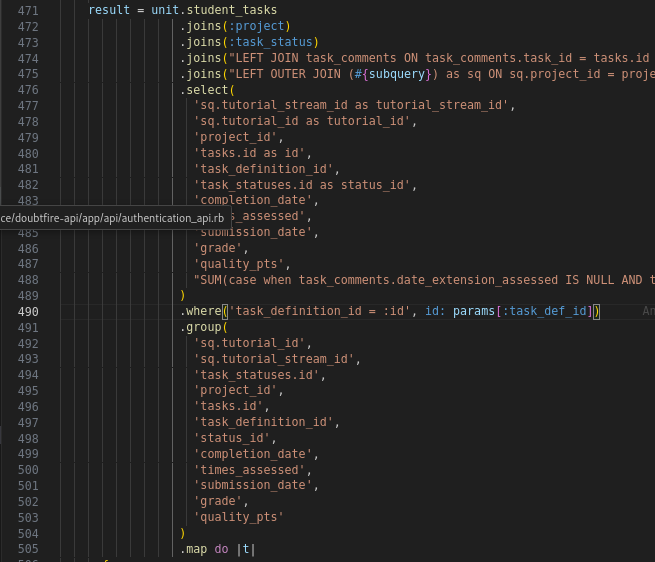
**Business Impact**

An SQL injection is quite harmful as it could lead to unauthorized access to the data, which could further lead to potential data leak and could potentially lead to the loss of confidence among the users. Moreover, it could lead to stealing of credentials, which could lead to possible breaches of data and violations of privacy. Hence, this vulnerability could lead to a major setback to the business as a whole.

**Affected Assets**

The assets that could be affected by the given vulnerable part of code is the main database in which the data is stored, this could lead to lack of Data integrity and confidentiality as the credentials of the users could be directly accessed by the attacker. The database server could be overwhelmed by the possibility of Denial-of-Service attack.

**Evidence**

****

**The set given code on the line 471 and 474 in specific in the file doubtfire\_deploy/doubtfire-api/app/api/task\_definitions\_api.rb**

**The vulnerable line in the above code is:**

**.joins("LEFT OUTER JOIN (#{subquery}) as sq ON sq.project\_id = projects.id")**

**In which the subquery is interpolated directly into the string without proper sanitization of the input.**

**Remediation Advice**

The following methodologies be used to make the code more secure and to remove the vulnerability:

* Use of parameterized queries in which we could use the clause to\_i to convert the id to integer before passing it into the query as follows:

**.where('task\_definition\_id = :id', id: params[:task\_def\_id].to\_i)**

* Use of Left\_joins**,** which would help to avoid direct SQL interpolation as:

**.left\_joins(:task\_comments)**

**.where("task\_comments.task\_id = tasks.id AND (task\_comments.type IS NULL OR task\_comments.type <> ?)", 'TaskStatusComment')**

This would allow to safely bind the string ‘TaskStatusComment' without direct interpolation and would certainly reduce the risk of SQL injection.

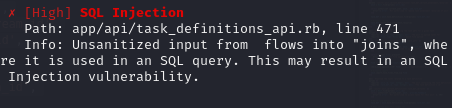
* Use of parameter binding such as ‘?’ wherever possible to reduce the risk of potential SQL injection in the code.

**References**

[**https://owasp.org/www-community/attacks/SQL\_Injection**](https://owasp.org/www-community/attacks/SQL_Injection)

[**https://www.w3schools.com/sql/sql\_injection.asp**](https://www.w3schools.com/sql/sql_injection.asp)

[**https://learn.snyk.io/lesson/sql-injection/**](https://learn.snyk.io/lesson/sql-injection/)

**In particular, to verify the vulnerability, I used the snyk CLI tool, for which the screenshot is given below:**

**Contact Details**

Gaurish Bhatia

S222187151@deakin.edu.au

**Pentest Leader Feedback.**

The lead will provide feedback to enact on.