NoSQL Injection

Web App Vulnerabilities

SECURE CODE WARRIOR

What is it?

NoSQL injections occur when an attacker is able to inject malicious code in a NoSQL database. This is typically done through JSON and Javascript code

What causes it?

The application makes use of a NoSQL database and does not properly validate user input before using it to query the database.

What could happen?

A succesful injection could allow an attacker to update, insert, or delete data. All data could be exposed or deleted. Access to the hosting system could be gained. Authentication could be bypassed.

How to prevent it?

Input should be JavaScript escaped or validated before being used to query the database. Consider all user-controllable input, including HTTP headers. Backend DB users should run with the least required privileges.

NoSQL Injection

Understanding the security vulnerability

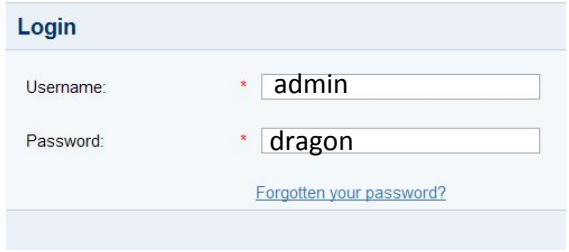
Scenario 1: Normal authentication workflow

A user submits his credentials using POST parameters.

The parameters are appended to a database query string that is submitted to the database.

The credentials are valid and the appropriate record is returned to the web server.

The session cookie is returned to the browser; the user is now logged in.



Client

{ “Username”: “admin”, “Password”: “dragon” }

Set-cookie: sessionid= FUHOJFB0I4BW121X7281

Web Server

db.users.find({username: ‘admin’, password: ‘dragon’});

Returned 1 Row (‘admin’, ‘John’, ‘Doe’)

DB Server

NoSQL Injection

Understanding the security vulnerability

Scenario 2: Authentication bypass

An attacker submits input values (through an interception proxy) that change the logic of the query. A NoSQL comparison operator is passed instead of the normal username and password.

The username and password from the database will be compared to the empty string resulting in an always true condition.

As a consequence of the always true comparison, the attacker is logged in as the first user in the database, the admin in this case.

The session cookie is returned to the browser; the attacker is now logged in as administrator.

NoSQL Injection

Realizing the impact

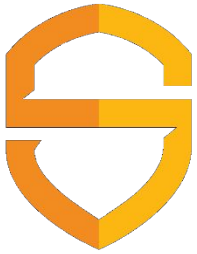
Altered data such as balance and transaction information could cause repudiation issues.

Account and private data theft could damage reputation and credibility, causing customer and revenue loss.

System unavailability could cause revenue and reputation loss.

NoSQL Injection

Preventing the mistake



Apply **white-list validation** on all user input.
Consider GET and POST parameters, Cookies and other HTTP headers.

Use a **database ORM** instead of doing raw queries.

Sanitise and **validate** input parameters.
coerce input to the correct types during validation routines

Apply the **least privilege** principle on the database users.