Second Dose

Problem Type

SQL Injection, Privilege Escalation, Authentication Bypass, Blacklist Bypass

---

Solution

Investigation

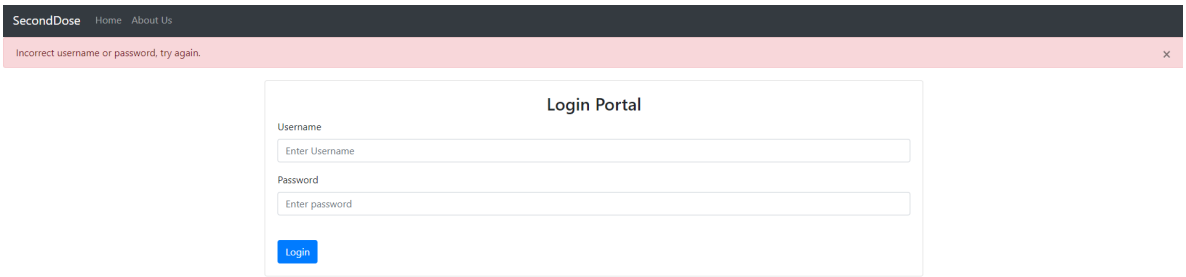
First, we look through the contents of the webapp. Nothing much seems of value on the home page, nor the 'about us' page.

We inspect the source code and we see that there is a login page that has been commented out on the navbar:

```
<div class="collapse navbar-collapse" id="navbar">
        <div class="navbar-nav">
          <!-- <a class="nav-item nav-link" id="login" href="/login">Login</a> -->
          <a class="nav-item nav-link" id="home" href="/">Home</a>
          <a class="nav-item nav-link" id="About" href="/about">About Us</a>
        </div>
      </div>
```

We then proceed to visit `/login`, a login page which requires both a username and password to login. We try the usual credentials:

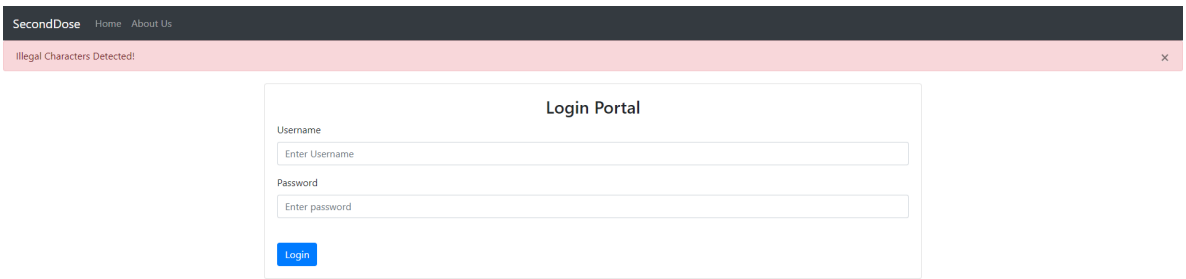
```
username: admin
password: admin
```



Unlike the challenge `first_dose`, the SQL injection query is not printed for us to view. We get an error: `Incorrect username or password ...`

We then suspect SQL Injection, and we inject the payload:

```
' OR 1=1;--
```



Unlike the first error alert we obtained, we get an error `Illegal Characters Detected!`. This shows us that the input field is vulnerable to SQL Injection, but we may have to modify our payload to bypass whatever blacklist the login page employs.

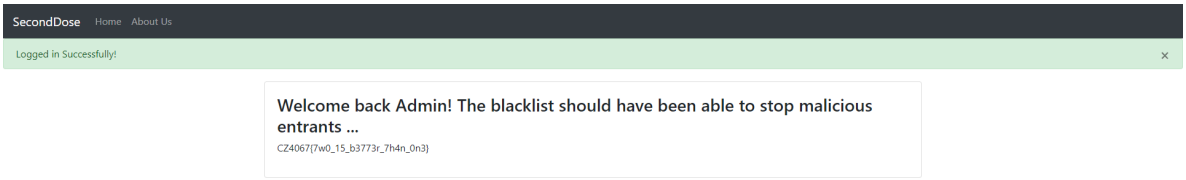
Payload

Through enumeration, we notice that there are many SQL Statements/Queries/Keywords that are blacklisted. In order to beat this blacklist, we ask ourselves: How is this blacklist implemented on the server?

One common blacklist method would be to remove one instance of the blacklisted word from the string. This can be bypassed by concatenating the keyword twice. We then try this newly crafted payload:

```
' OORR 1=1;--
```

We bypass the filter and get the flag!



Flag

`CZ4067{7w0_15_b3773r_7h4n_0n3}`