1. **Broken Access Control:**

Broken access control vulnerabilities exist when a user can in fact access some resource or perform some action that they are not supposed to be able to access.

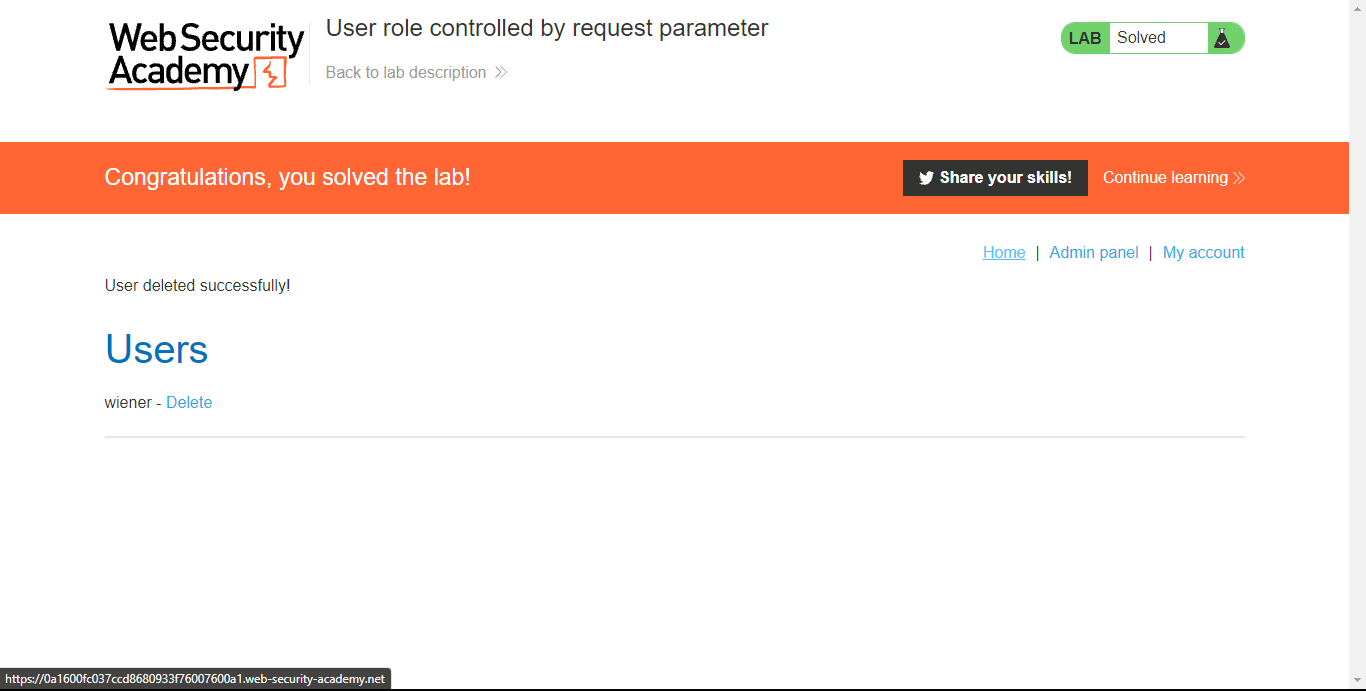
Some applications determine the user's access rights or role at login, and then store this information in a user-controllable location, such as a hidden field, cookie, or preset query string parameter. The application makes subsequent access control decisions based on the submitted value.

This approach is fundamentally insecure because users can modify the value and gain access to functionality they are not authorized to, such as administrative functions.

This lab has an admin panel at /admin, which identifies administrators using a forgeable cookie.

Solve the lab by accessing the admin panel and using it to delete the user carlos.

You can log in to your own account using the following credentials: wiener:peter.



1. **Injection:**

Injection is an attacker’s attempt to send data to an application in a way that will change the meaning of commands being sent to an interpreter. For example, the most common example is SQL injection, where an attacker sends “101 OR 1=1” instead of just “101”.

There are lots of interpreters in the typical web environment, such as SQL, LDAP, Operating System, XPath, XQuery, Expression Language, and many more. Anything with a “command interface” that combines data into a command is susceptible. Even XSS is really just a form of HTML injection.

This lab contains an OS command injection vulnerability in the product stock checker.

The application executes a shell command containing user-supplied product and store IDs, and returns the raw output from the command in its response.

To solve the lab, execute the whoami command to determine the name of the current user.



