UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

PortSwigger Academy

PRACTICAL WORK 3

TIZIANO NARDONE
UNIVERSITY OF REIMS | FRANCE

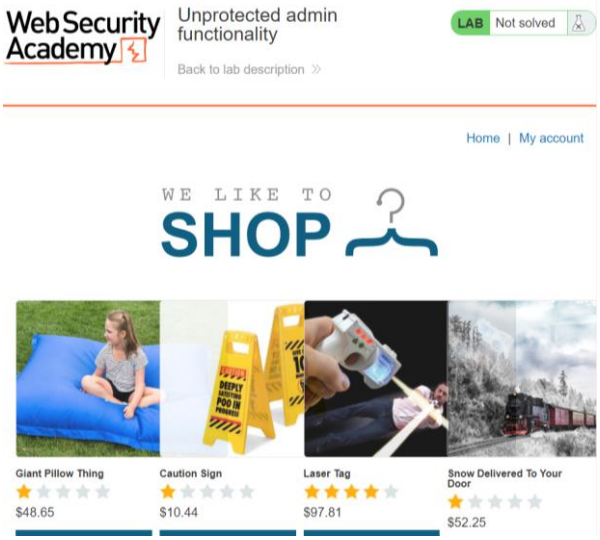
Contents

I. Access control vulnerabilities

1. Unprotected admin functionality

Context :

This lab has an unprotected admin panel and to solve the lab, ***carlos*** user has to be deleted. We are playing the role of a black hat threat. Indeed, we apparently just have a simple access to the retail website and don't have any credentials.



Vulnerability :

To delete ***carlos*** user we must gain access to an administrative page or functionality. This is ***Vertical Privilege Escalation***.

Exploitation :

An administrative web page might be accessed by directly browsing the relevant admin URL.

For instance we might type the following URL :

https://acc91f931e0c5dfbc06f3930009c00fd.web-security-academy.net***/admin***

However the web page doesn't exist :

```
"Not Found"
```

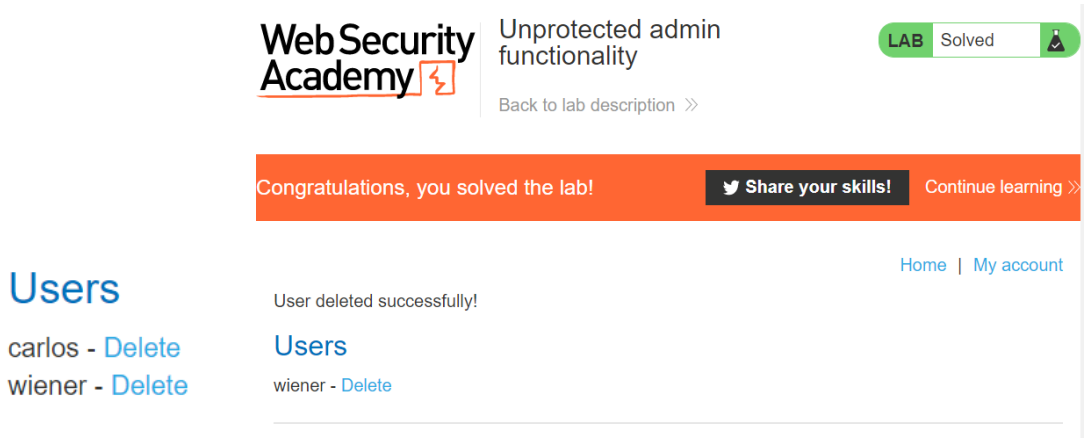
Furthermore, in some cases the administrative web page might be located in another file like ***robots.txt*** like suggested on the access control course :

https://acc91f931e0c5dfbc06f3930009c00fd.web-security-academy.net***/robots.txt***

```
User-agent: *
Disallow: /administrator-panel
```

That result gives us the information of the location of the admin panel. Thus, by tapping the following URL, we've gained access to the admin page and we've could delete ***carlos*** user.

https://acc91f931e0c5dfbc06f3930009c00fd.web-security-academy.net***/administrator-panel***



Advises :

It is not recommended to choose a common name for URIs which gain access to sensitive functionalities.

Even if the URL isn't disclosed anywhere, a threat may be able to use a wordlist to brute-force the location of the sensitive functionality. So any sensitive functionalities must be protected by password policies with even multiple-factor authentication. Plus it might be interesting to specialized users and groups minimum privileges.

2. Unprotected admin functionality with unpredictable URL

Context:

This lab has an unprotected admin panel. It's located at an ***unpredictable location***, but the location is disclosed somewhere in the application. Once again we must solve the lab by accessing the admin panel and using sensitive functionalities to delete the user ***carlos***.

Vulnerability:

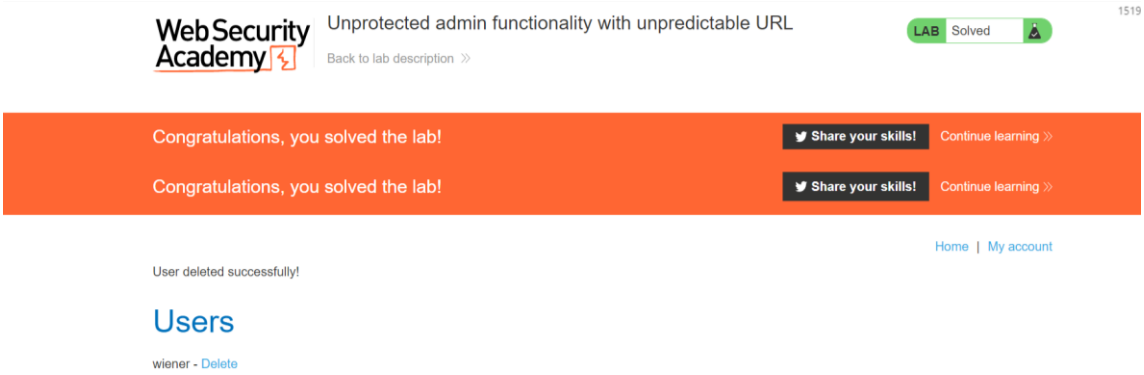
To delete ***carlos*** user we still must gain access to an administrative page or functionality. This is another example of ***Vertical Privilege Escalation.*** The difference is that all common URIs of admin panel location don't work.

Exploitation:

By inspecting the source web page, we noticed that there is a JavaScript code which check if the current user of the source page is an admin or not and if it does, the script creates a link to give access to the user to reach the admin panel. Thus we found the URI of the admin panel.

```
<script>
  var isAdmin = false;
  if (isAdmin) {
     var topLinksTag = document.getElementsByClassName("top-links")[0];
     var adminPanelTag = document.createElement('a');
     adminPanelTag.setAttribute('href', '/admin-egd5tf');
     adminPanelTag.innerText = 'Admin panel';
     topLinksTag.append(adminPanelTag);
     var pTag = document.createElement('p');
     pTag.innerText = '|';
     topLinksTag.appendChild(pTag);
  }
   == $0
</script>
```

So by tapping the following the URL we gained access to the admin panel and deleted the ***carlos*** user.



Advises:

It's not recommended to disclose the URL of admin pages of functionalities within JavaScript code. Plus we may add the same recommendations as the previous to make the URI admin panel not guessable.

3. User role controlled by request parameter

Context:

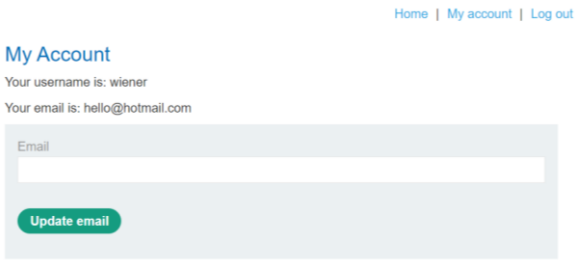
To solve this lab we must still gain access to the admin panel whose the URI is ***/admin*** and delete the user ***carlos***. We can log in to our own account using the following credentials : wiener:peter

Vulnerability:

To delete ***carlos*** user we still must gain access to an administrative page or functionality. This is another example of ***Vertical Privilege Escalation.*** In this case the difference is that we know the URI of the admin panel but something checks if the user is an admin or not.

Exploitation:

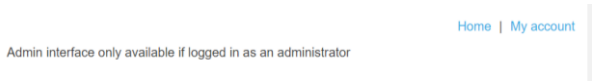
Firstly, we logged in to our own account.



Then we tried to gain access to the admin panel by typing the following URL:

https://acc61fbe1f249e77c09a2bec007b00e5.web-security-academy.net***/admin***

But the problem was that we didn't get the access because we weren't an admin user.

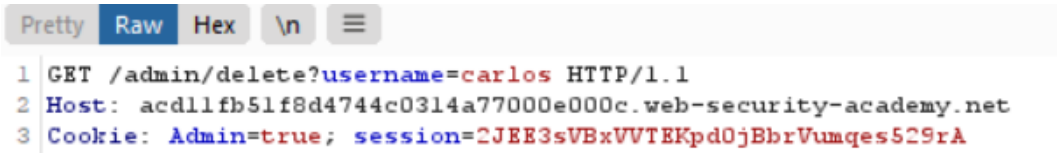


To gain access to the admin panel we used the Burp proxy, intercepted the ***GET*** request and changed the cookie ***Admin*** from ***false*** to ***true***.

```
GET /admin HTTP/1.1
Host: acc61fbe1f249e77c09a2bec007b00e5.web-security-academy.net
Cookie: Admin=true; session=Nj6vCmxWpMRDtQvfAMM2dMsi2QAqxXvG
```

Then we gained access to the admin panel.

We repeated the same action to delete the ***carlos*** user.



Once again to see the result.

Advises:

It is not recommended to used to cookie as an authentication process. Thus, as written previously it is more recommended to permit the access of those sensitive functionalities with multiple-factor authentication.

II. Business logic vulnerabilities

1. Excessive trust in client-side controls

Context:

To solve the lab we must complete a purchase workflow of the “lightweight Leather jacket” item. The problem is that we don’t have enough money on our account to complete the purchase.

Vulnerability:

To complete the purchase we must find the vulnerability related to the price of the item.

Exploitation:

While we were trying to complete the purchase even though we didn’t have enough, to see what would go on, we found that at the moment to send a request to add the item to the shopping cart that we were sending the price of the item through a POST request.

```
productId=1&redir=PRODUCT&quantity=1&price=133700
```

By reducing the price to a lower amount than our store credit,

```
productId=1&redir=PRODUCT&quantity=1&price=10
```

We finally succeeded to complete the purchase.

Total: $0.10

Advises:

It's not recommended to pass any sensitive information by users, we would prefer send the id of the item and ask for its price to a secured database so as to remain the transaction hidden to users (clients).

2. High-level logic vulnerability

Context:

Once again, to solve the lab we must complete a purchase workflow of the "lightweight Leather jacket" item. The problem is that we don't have enough money on our account to complete the purchase

Vulnerability:

To complete the purchase we must intercept requests and responses to see if we could change any sensitive data.

Exploitation:

By intercepting a request to add the item to the shopping cart, we see that this time we can change the quantity of the but not the price.

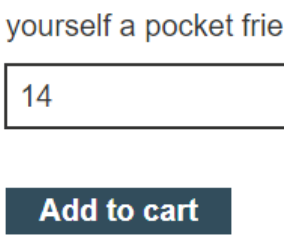
```
productId=1&redir=PRODUCT&quantity=1
```

After trying multiple random quantities, we realised that setting a negative quantity worked.

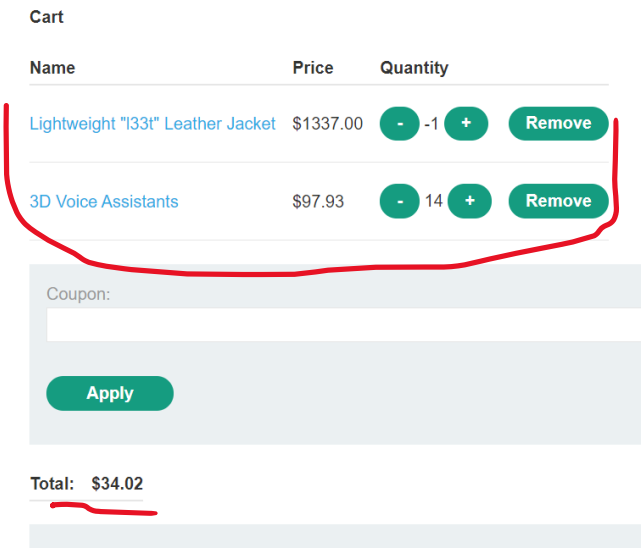
```
productId=1&redir=PRODUCT&quantity=-1
```

Total: -$1337.00

Afterwards, we added 14 3D voice assistants of 97$ each to the cart but it didn't work and the amount printed was above of our store.



So we tried the opposite way by adding 14 3D voices assistants first and a negative quantity for the leather jacket and it worked, we had enough money to complete the purchase.



Advises:

Obviously we should check for the quantity of item to make sure that it won't be negative. Plus we should be careful and prevent code injections by escaping characters or encoding them.

3. Weak isolation on dual-use endpoint

Context:

To solve this lab, we must get access to the administrator account and delete the user Carlos.

Vulnerability:

As it is a business logic vulnerability, we must exploit the logic of resquests/reponses to get the access.

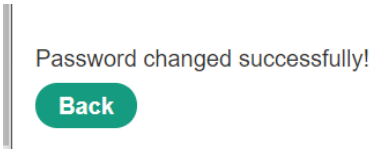
Exploitation:

Once we logged in, we saw that we could change the current password of our user.

```
4&username=wiener&current-password=peter&new-password-1=azerty&new-password-2=azerty
```

After multiple tests we noticed that we could change the current password of our user without sending the attribute "current-password".

```
4&username=wiener&current-password=peter&new-password-1=azerty&new-password-2=azerty
```



As we could change the password of our user without specify it in attributes, we changed the username to “administrator” and changed his password to get access to the admin panel.

```
username=administrator&new-password-1=azerty&new-password-2=azerty
```

Advises:

It’s recommend to be less clear for requests/responses logic, check if every attributes are presents and prevent code injections.