DATE: 2024-12-19
INFORMATION CLASS: CUSTOMER CONFIDENTIAL

CUSTOMER	SHOP AND BLOG
SUBJECT	WEB APPLICATION
DOCUMENT	SECURITY ASSESSMENT REPORT

Pentest AB
ORG NR. 12332112332
WWW.pentestab.COM

Table of contents

1. EXECUTIVE SUMMARY

During the period between 2024-12-09 and 2024-12-19, Pentest AB conducted a security assessment of Shop and Blog's customer account portal, blog page and web shop.

The purpose of this assessment was to evaluate the current security status of the blog page, web shop and account portal. There was also focus on the availability of user input on these items listed above.

Shop and Blog handles customers personal and financial data, therefore the security assessment aimed to identify potential attacks that attackers could exploit to access or leak customer data or access certain elements within the web application that users should not be able to access. This report presents the findings of the assessment, providing technical details about the identified vulnerabilities along with recommendations for their mitigation.

1.1 Results

Vulnerabilities within the platform were identified by the security assessment. These vulnerabilities could potentially allow unauthorized access to certain functionalities and customer data. By manipulating certain elements within these vulnerabilities, attackers could gain access to sensitive customer information, send malicious code via the website that could attack customers visiting the website, and manipulate prices on items listed in the shop.

The security assessment revealed three instances of vulnerabilities where each instance belongs to a different category. The primary issue that was found, and tied together these vulnerabilities, was trust in user input to the application.

1.2 Recommendations

Make an effort to mitigate all the vulnerabilities listed in this security assessment, even the one with a lower severity. Though there are not any identified vulnerabilities in this assessment with a low severity, mitigating possible future vulnerabilities with a low severity increases the security posture of the application. Mitigating low severity issues can be important because attackers can chain several low severity issues, and therefore create a high severity issue. By mitigating low severity issues the chain can break and prevent a large attack.

This security assessment can be used in the future if Shop and Blog decides to update or expand their application. Take these vulnerabilities that has been found and check with the developers if they have implemented the proper security for these issues. If there are new developers, perhaps educate them on these vulnerabilities.

2. FINDINGS AND RECOMMENDATIONS

This section explains the approach in which the testing was conducted during the security assessment. This section also provides recommendations on improving the application's security posture based on the found vulnerabilities. More detailed information about found vulnerabilities can be found in chapter 3, subchapter 3.3 *Technical description of findings*.

2.1 Approach to Testing

The performed security assessment was a web application assessment, and the goal of this assessment was to identify vulnerabilities such as misconfigurations, weaknesses and technical flaws in the web application. The process to do such an assessment involves certain tools and frameworks, such as the OWASP Top 10:2021 and OWASP Web Security frameworks. These frameworks are well known for aiding in testing in web applications and has been used in aiding this assessment for analyzing security mechanisms and how the application works. The tools that were used during the assessment was mainly Burp Suite.

Security testing is not an exact science, and it is not possible to list every single test-case that can be performed against an application, but some common areas are:

- Authorization
- Injection attacks
- Error handling
- Cryptography
- Business logic

The approach to finding these vulnerabilities was to test them manually using the web application and Burp Suite. To find vulnerabilities in the account portal the assessors was given test accounts provided by Shop and Blog.

2.2 Findings and Recommendations

The evaluation of the application showed multiple locations that had vulnerabilities that an attacker could exploit. Firstly, the user blog page has multiple locations, more specifically the comment function, that are susceptible to injection attacks, specifically Cross-Site Scripting (XXS). Due to improper input validation mechanisms in the comment function attackers can execute malicious scripts that can harm unsuspecting users. Since the vulnerability resides in the blog page, a user would be attacked by clicking on a blog post that has been infected. Vulnerabilities such as this one could compromise the integrity and confidentiality of user data and potentially sensitive information.

To mitigate the risk associated with identified XSS vulnerabilities, it is recommended to filter user input on arrival and encode data in output. It is important to validate input, to filter as strictly as possible based on what is expected or valid input. By applying input validation and data encoding throughout the application, specifically in the comment section of the blog page, but also in other areas of the application, the identified XSS vulnerabilities could be mitigated.

The second finding applies to the user account portal, where a vulnerability in access control exists. Due to improper input mechanisms in the URL-bar and the possibility to view specific data in the application's response to changes to the URL, it is possible for an attacker to obtain the administrator's password.

To mitigate the risk associated with identified access control vulnerabilities, it is recommended to deny access by default, unless a resource is intended to be publicly accessible. It is also recommended to declare the access that is allowed for each resource at code level and deny access by default, and then thoroughly audit and test access controls to ensure they work as designed. By addressing the root cause of what is allowed to be typed into the application's URL and what data response the application gives, and applying the recommended actions, the identified access control vulnerability could be mitigated.

The third finding applies to the application's web shop, specifically the "cart", where a business logic vulnerability exists. Due to possible flawed assumptions about user interaction with the application and the possibility to intercept specific data in the application's response, it is possible for an attacker to purchase items without paying full price.

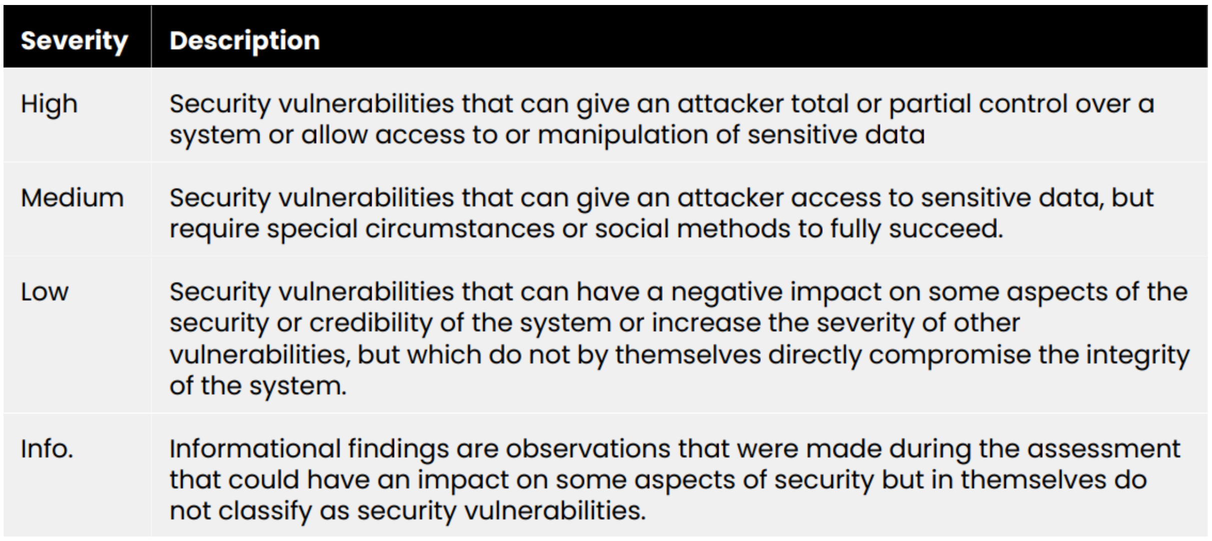
To mitigate the risk associated with identified business logic vulnerabilities, it is recommended to have a comprehensive approach to understanding how the application react in different scenarios, and to avoid making implicit assumptions about user interaction. By not trusting the users, understanding the application and its domain, and controlling what data response the application gives, the identified business logic vulnerability could be mitigated.

2.3 Delimitations and Restrictions

While source code and test accounts were provided, the test accounts were only for regular users, no administrative accounts were provided.

3. RESULTS AND RECOMMENDATIONS

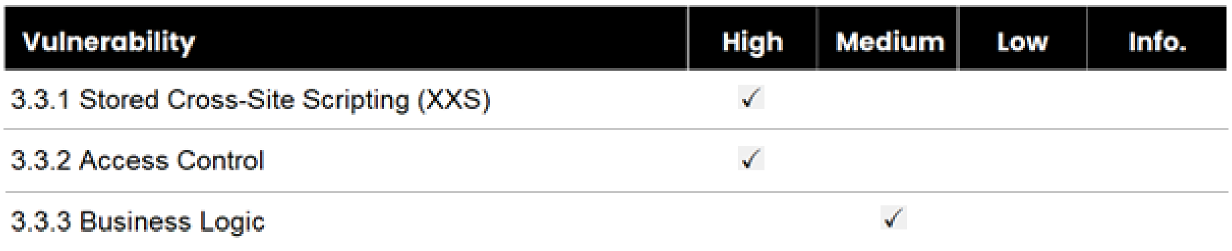
3.1 Severity ratings



Severity	Description
High	Security vulnerabilities that can give an attacker total or partial control over a system or allow access to or manipulation of sensitive data
Medium	Security vulnerabilities that can give an attacker access to sensitive data, but require special circumstances or social methods to fully succeed.
Low	Security vulnerabilities that can have a negative impact on some aspects of the security or credibility of the system or increase the severity of other vulnerabilities, but which do not by themselves directly compromise the integrity of the system.
Info.	Informational findings are observations that were made during the assessment that could have an impact on some aspects of security but in themselves do not classify as security vulnerabilities.

Table 1: Severity ratings.

3.2 Outline of identified vulnerabilities



Vulnerability	High	Medium	Low	Info.
3.3.1 Stored Cross-Site Scripting (XXS)	✓			
3.3.2 Access Control	✓			
3.3.3 Business Logic		✓		

Table 2: Identified vulnerabilities.

3.3 Technical description of findings

3.3.1 Stored Cross-Site Scripting (XXS)

Severity: high

Description

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected by an attacker into, in this context, a vulnerable application. An attacker executing an XSS attack can hijack the user's session, acting within the application as the affected user. XXS works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

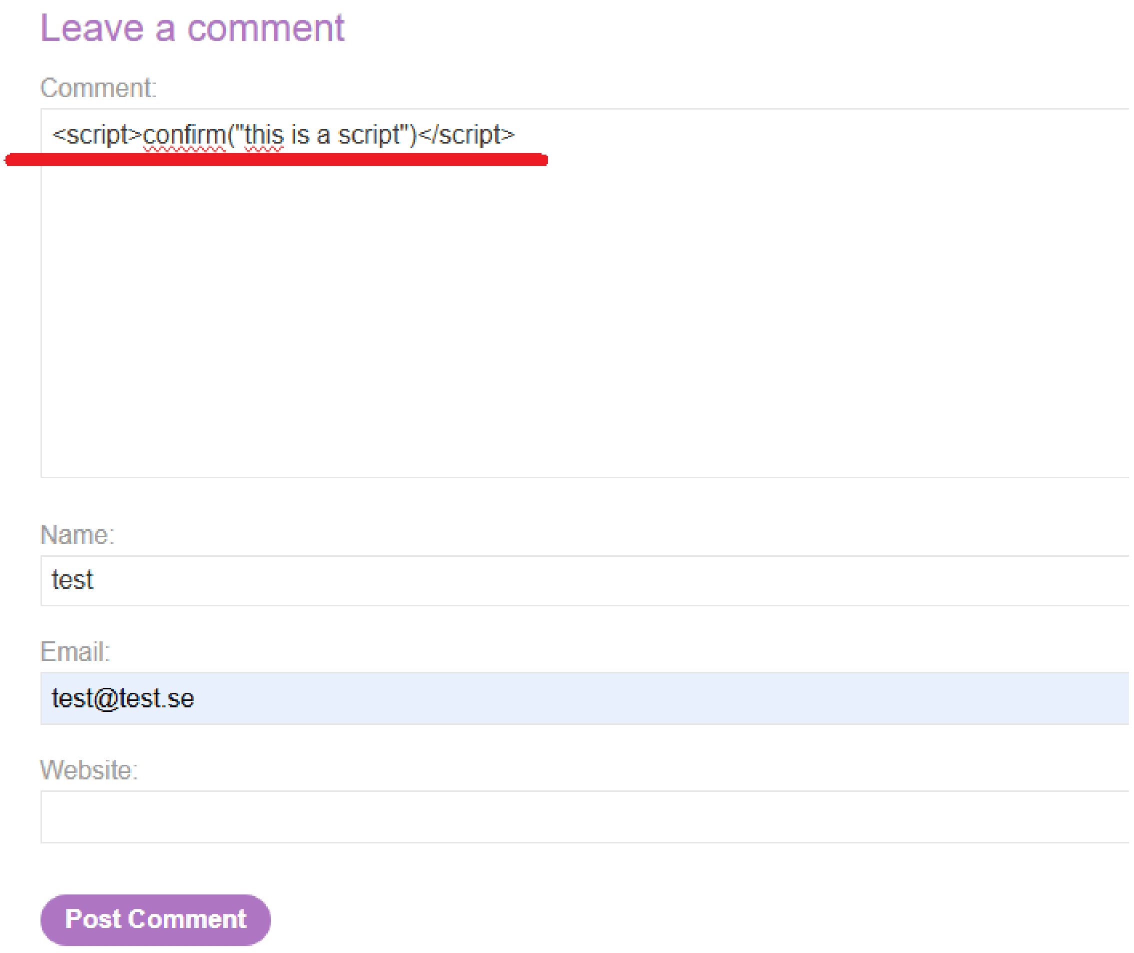
There are several types of XSS attacks, and during the assessment it was discovered that the application was vulnerable to a stored XSS attack. Stored XXS is when the injected code is stored in the back-end, such as in a message forum, visitor log, comment field, etc. Later, the stored code will be included in the response sent to a visiting user. Stored XSS is generally easier to utilize by an attacker, as it may not involve any extra user interaction by the victim.

The application was susceptible to stored XXS due to its handling of user input. Since the application did not properly validate user input and not properly encode certain characters used in forming the page code, JavaScript can be used by an attacker. Therefore, JavaScript can be executed in an unsuspecting user's web browser, and in this instance when a user is visiting a blog post.

During the security assessment it was discovered that all the posts within the shopandblog.com/post?postId=x was vulnerable to stored XXS.

The security assessment showed that the comment section is susceptible to injections of JavaScript, specifically where the user can write a comment. As a result, an attacker could exploit this to target blog post users.

In image 1 below, a screenshot of a blog post can be seen, where the comment section contains a script that is as follows: **<script>confirm("this is a script")</script>**. The script calls for a pop-up window to appear in the application, and if the window appears it means that the application has accepted the script, and in this assessment a window appeared. This means that JavaScript in a victim's browser was executed when they navigated to the blog post. Since this vulnerability exists in all blog posts, it means that a user is vulnerable when they visit any of the posts, assumed that an attacker has injected malicious code on all posts.



Leave a comment

Comment:
<script>confirm("this is a script")</script>

Name:
test

Email:
test@test.se

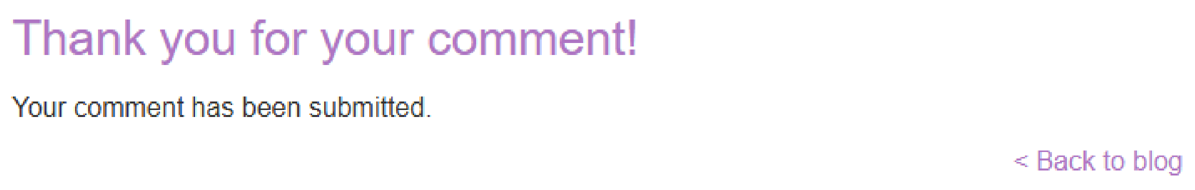
Website:

Image 1: Script tag within comment section

The example below shows the POST request containing the JavaScript that was inserted into the application.

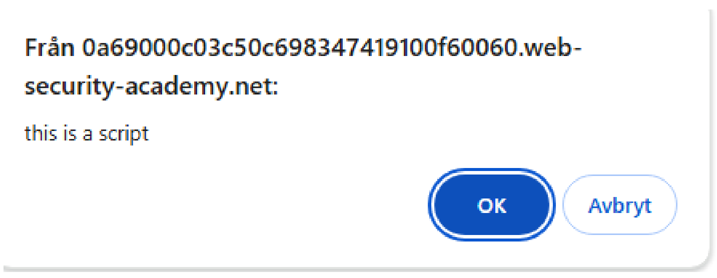
```
POST /post/comment HTTP/2
Host: 0ad4008104309c6f87c413110036004e.web-security-academy.net
Cookie: [...]
Content-Length: 169
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
csrf=cb33[...]19GP&postId=7&comment=%3Cscript%3Econfirm%28%E2%80%9Cthis+is+a+script
%E2%80%9D%29%3C%2Fscript%3E&name=test&email=test%40test.se&website=
```

Example 1: POST request


Thank you for your comment!

Your comment has been submitted.

Image 2: Comment submitted


Från 0a69000c03c50c698347419100f60060.web-security-academy.net:

this is a script

Image 3: Window confirming the script was executed, when returning to blog post

Recommendations

Data supplied by a user should be seen as untrusted data and should be validated and encoded according to the context in which it is included, to prevent it from being treated as part of the page code structure.

For more information on Cross-Site Scripting mitigation, see:

OWASP Cheat Sheet Series – Cross Site Scripting Prevention Cheat Sheet[1]

[1] https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

3.3.2 Broken Access Control

Severity: high

Description

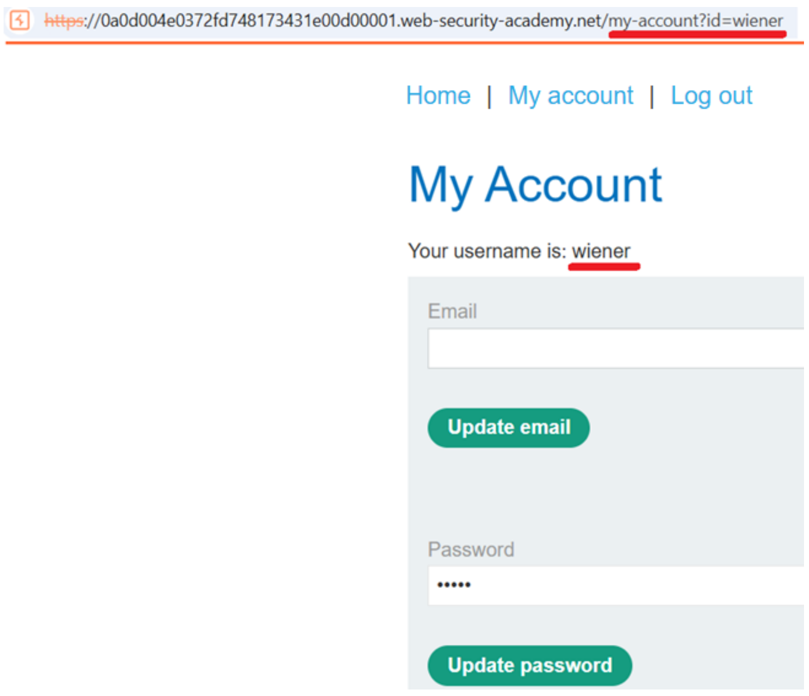
Access control is the application of constraints on who or what is authorized to perform actions or access resources. In the context of web applications, access control is dependent on authentication and session management. When an application has a broken access control, attackers could exploit this to gain access to certain areas of an application that should normally be out of reach. Broken access control poses a critical security vulnerability since users with restricted access, for example, could access data that is normally restricted to them, such as administrator privileges.

Pentest AB identified broken access control in the "id" parameter in the URL when logged-in to the user account page. This made it possible to access the administrator password, and therefore administrator privileges, such as deleting users.

The security assessment showed that by logging in to the user account page using the supplied account credentials by Shop and Blog, it was possible to change the "id" parameter in the URL to "administrator". By doing so it was possible to view the administrator's account, but not gain administrative privileges. When changing the "id" parameter and viewing the application's response in Burp Suite, it was possible to access the administrative password written within the HTML code. By exploiting this vulnerability an attacker could log-in to the administrator's account using "administrator" as username and the found password in the application's response, to make changes to the application, such as deleting users.

The vulnerability was verified by logging in to the administrative account using this vulnerability and deleting a test user with the name "carlos".

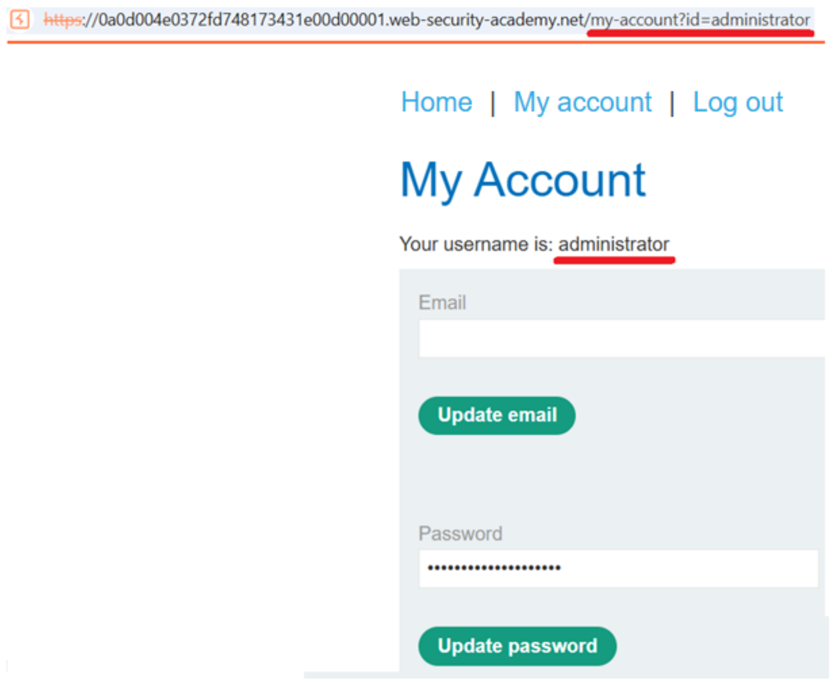
In the image below, a screenshot of the user account page can be seen, where the logged-in user is “wiener”, and the “id” parameter in the URL is set to “wiener”.


~~https~~://0a0d004e0372fd748173431e00d00001.web-security-academy.net/my-account?id=wiener

Your username is: wiener

Image 4: Regular user account page

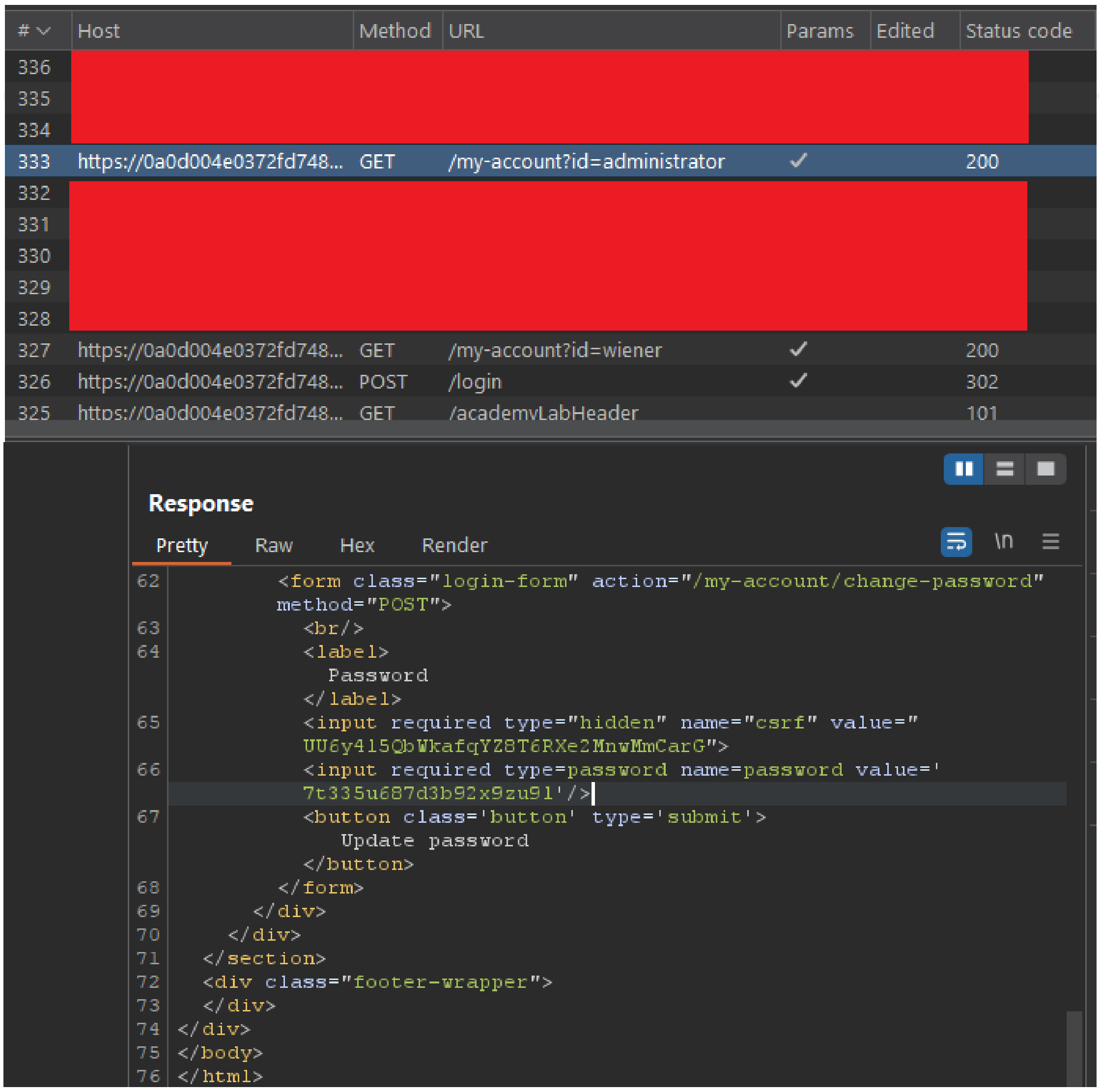
The image below shows the administrator account page when changing the “id” parameter in the URL to “administrator”.


~~https~~://0a0d004e0372fd748173431e00d00001.web-security-academy.net/my-account?id=administrator

Your username is: administrator

Image 5: Administrator account page without administrative privileges

The image below shows the application’s response when changing the URL id to “administrator”. On line 66 in the HTML code the administrative password can be seen.



#	Host	Method	URL	Params	Edited	Status code
336	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
335	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
334	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
333	https://0a0d004e0372fd748...	GET	/my-account?id=administrator	✓		200
332	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
331	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
330	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
329	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
328	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
327	https://0a0d004e0372fd748...	GET	/my-account?id=wiener	✓		200
326	https://0a0d004e0372fd748...	POST	/login	✓		302
325	https://0a0d004e0372fd748...	GET	/academyLabHeader			101

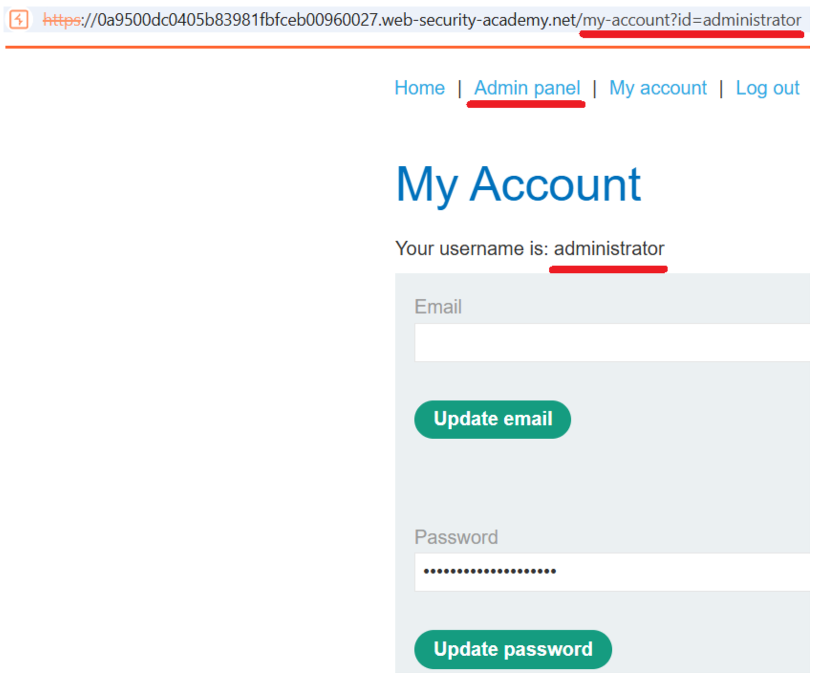
```
62          <form class="login-form" action="/my-account/change-password"
            method="POST">
63            <br/>
64            <label>
                Password
              </label>
65            <input required type="hidden" name="csrf" value="
              UU6y415QbWkafqYZ8T6RXe2MnwMmCarG">
66            <input required type=password name=password value='
              7t335u687d3b92x9zu9l'/>
67            <button class='button' type='submit'>
                 Update password
              </button>
68          </form>
69        </div>
70      </div>
71    </section>
72    <div class="footer-wrapper">
73    </div>
74  </div>
75  </body>
76  </html>
```

Image 6: Administrator’s password

```
<input required type=password name=password value='7t335u687d3b92x9zu9l'/>
```

Example 2: Administrator’s password, code example

The image below shows that it was possible to log-in to the administrator account using the found password. The image shows access to the “Admin panel”, which only administrators have access to.


~~https~~://0a9500dc0405b83981fbfceb00960027.web-security-academy.net/my-account?id=administrator

Home | Admin panel | My account | Log out

My Account

Your username is: administrator

Email

Update email

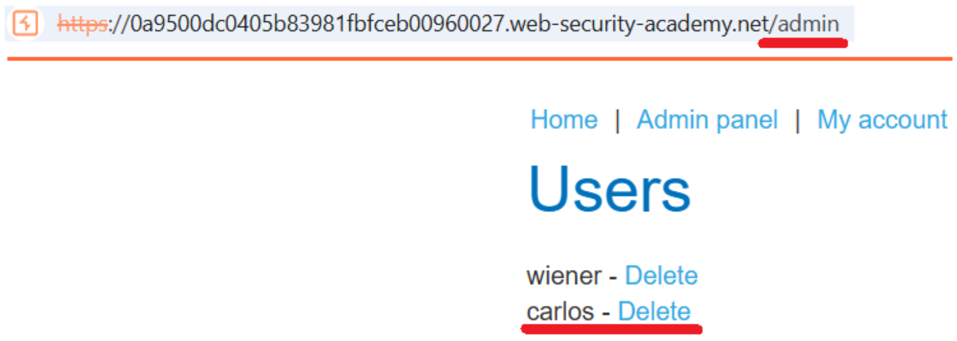
Password

••••••••••••••••••••

Update password

Image 7: Administrator account page with administrative privileges, broken access control

The image below shows the “Admin panel” and the ability to delete users.

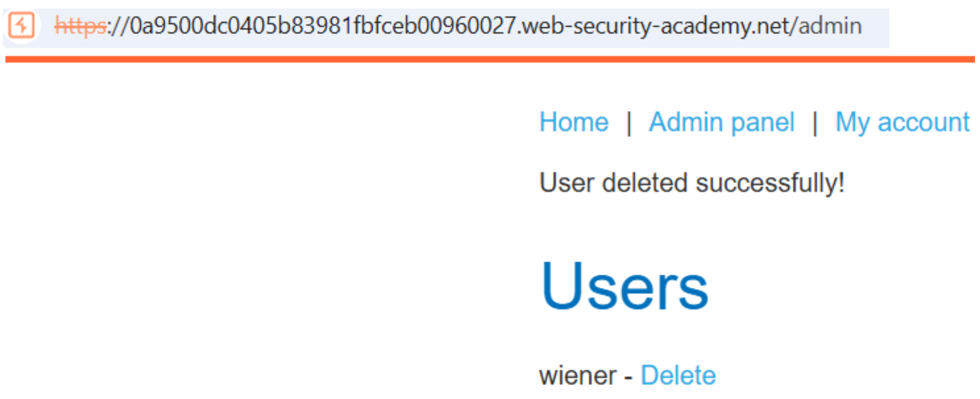

~~https~~://0a9500dc0405b83981fbfceb00960027.web-security-academy.net/admin

Home | Admin panel | My account

Users

wiener - Delete
carlos - Delete

Image 8: Admin panel


~~https~~://0a9500dc0405b83981fbfceb00960027.web-security-academy.net/admin

Home | Admin panel | My account

User deleted successfully!

Users

wiener - Delete

Image 9: Deleted user "carlos"

Recommendations

Address the root cause of what is allowed to be typed into the application's URL and what data response the application gives. Do not rely on data concealment alone for access control, and thereby do not underestimate users and what they can input into the application. Make sure to encode and declare access at code level, so when viewing the application's response, the password is not showing.

For more information on mitigating broken Access Control, see:

OWASP Cheat Sheet Series – Authorization Cheat Sheet[2]

[2] https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

3.3.3 Business Logic Vulnerability

Severity: medium

Description

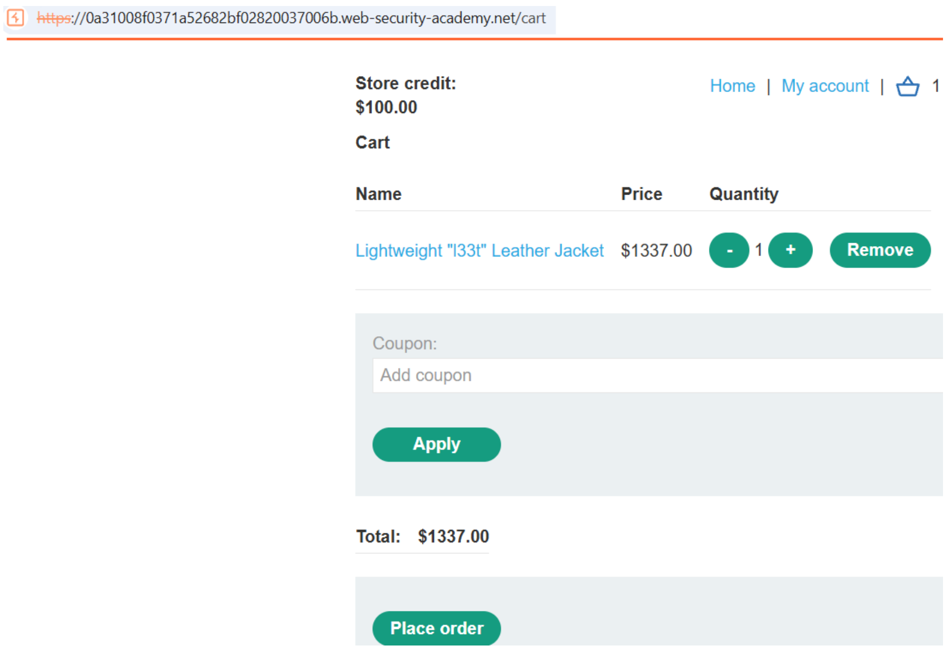
A vulnerability in business logic allows an attacker to elicit unintended behavior due to flaws and implementation of an application. This affects an application's legitimate functionality and could enable an attacker to manipulate such functionalities to achieve a malicious goal. Business logic vulnerabilities differ from most security problems. Most security problems are weaknesses in an application that result from a missing or broken security control. In turn, business logic vulnerabilities are ways of using the legitimate processing flow of an application in a way that results in a negative consequence to an application.

Pentest AB identified a business logic vulnerability in the web shop, specifically when adding an item to the "cart". This vulnerability made it possible to purchase an item for an unintended price.

The security assessment showed that by adding an item to the cart, and when adding another item to the cart, and changing the quantity of the second item to a negative number, it was possible to lower the total amount of money that the customer is supposed to pay. It was possible to change the quantity of an item by changing its parameter in the carts HTTP POST request using Burp Suite. For example, an attacker could add the item that they want to purchase to the cart, and then when adding the second item they could change its quantity parameter to a negative number. Thereby changing the second item's value to negative. By exploiting this vulnerability an attacker could purchase an item for an unintended price using another item from the web shop.

This vulnerability was verified by purchasing the "Lightweight l33t leather jacket" using the supplied credit of 100 dollars.

In the image below, a screenshot of the “cart” can be seen, with one “Lightweight l33t leather jacket”, worth $1337.00, added to it.



~~https~~://0a31008f0371a52682bf02820037006b.web-security-academy.net/cart

Store credit:
$100.00

Cart

Name	Price	Quantity
Lightweight "l33t" Leather Jacket	$1337.00	- 1 + Remove

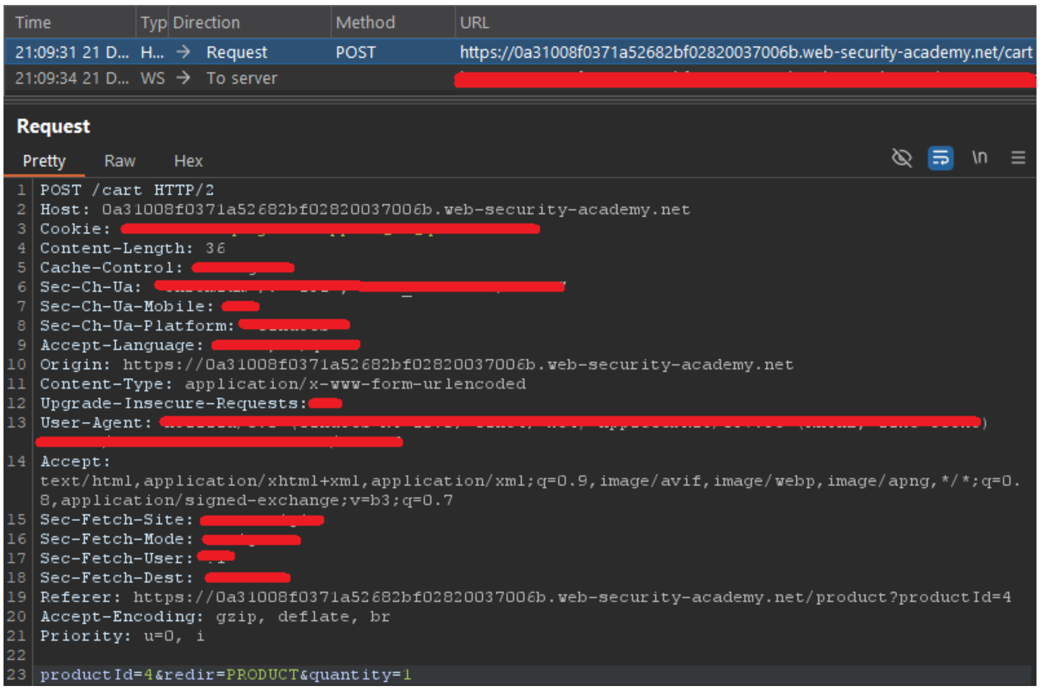
Coupon:
Add coupon
Apply

Total: $1337.00

Place order

Image 10: Cart containing the item that is intended to be bought

In the image below, a screenshot of an intercepted POST request can be seen. This is the request sent when adding the second item to the cart. This request shows the quantity parameter of the added item. The POST request was intercepted using Burp Suite.



Time	Typ	Direction	Method	URL
21:09:31 21 D...	H...	→ Request	POST	https://0a31008f0371a52682bf02820037006b.web-security-academy.net/cart
21:09:34 21 D...	WS	→ To server		[redacted]

Request

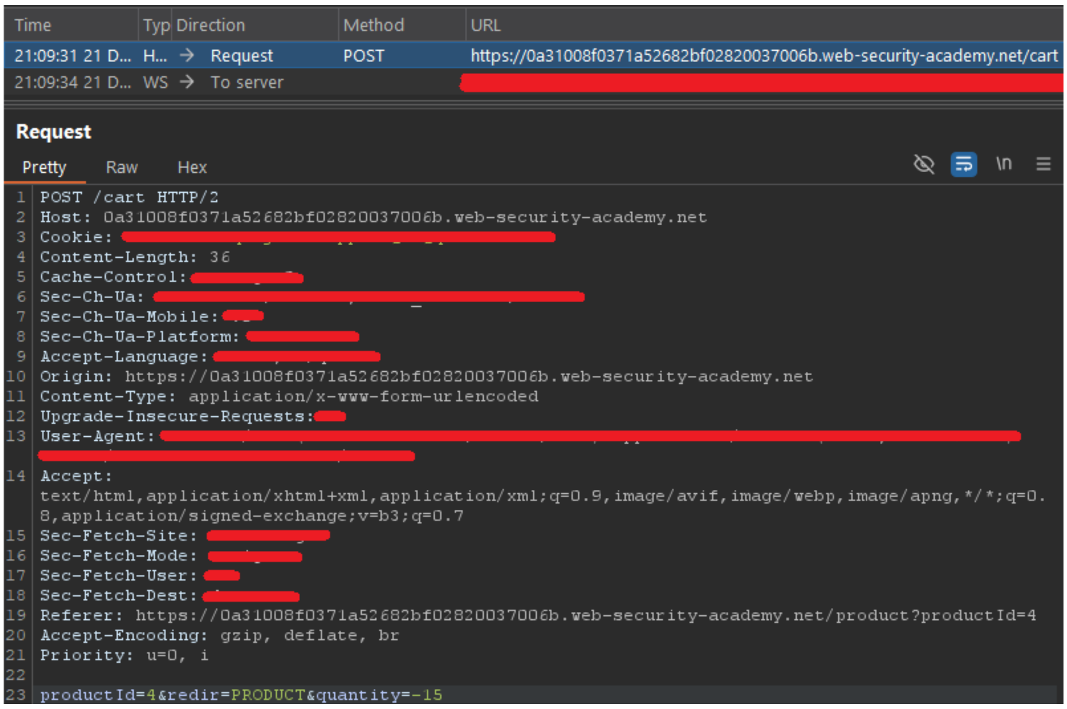
Pretty Raw Hex

```
POST /cart HTTP/2
Host: 0a31008f0371a52682bf02820037006b.web-security-academy.net
Cookie: [redacted]
Content-Length: 36
Cache-Control: [redacted]
Sec-Ch-Ua: [redacted]
Sec-Ch-Ua-Mobile: [redacted]
Sec-Ch-Ua-Platform: [redacted]
Accept-Language: [redacted]
Origin: https://0a31008f0371a52682bf02820037006b.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: [redacted]
User-Agent: [redacted])
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: [redacted]
Sec-Fetch-Mode: [redacted]
Sec-Fetch-User: [redacted]
Sec-Fetch-Dest: [redacted]
Referer: https://0a31008f0371a52682bf02820037006b.web-security-academy.net/product?productId=4
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

productId=4&redir=PRODUCT&quantity=1
```

Image 11: POST request with quantity=1

In the image below, a screenshot of the same POST request showed in image 11 can be seen, but with the quantity parameter on line 23 manually changed from 1 to -15. This is the location of the business logic vulnerability. It is possible to change the quantity parameter to a negative number, affecting the items value.



Time	Typ	Direction	Method	URL
21:09:31 21 D...	H...	→ Request	POST	https://0a31008f0371a52682bf02820037006b.web-security-academy.net/cart
21:09:34 21 D...	WS	→ To server		[redacted]

Request

Pretty Raw Hex

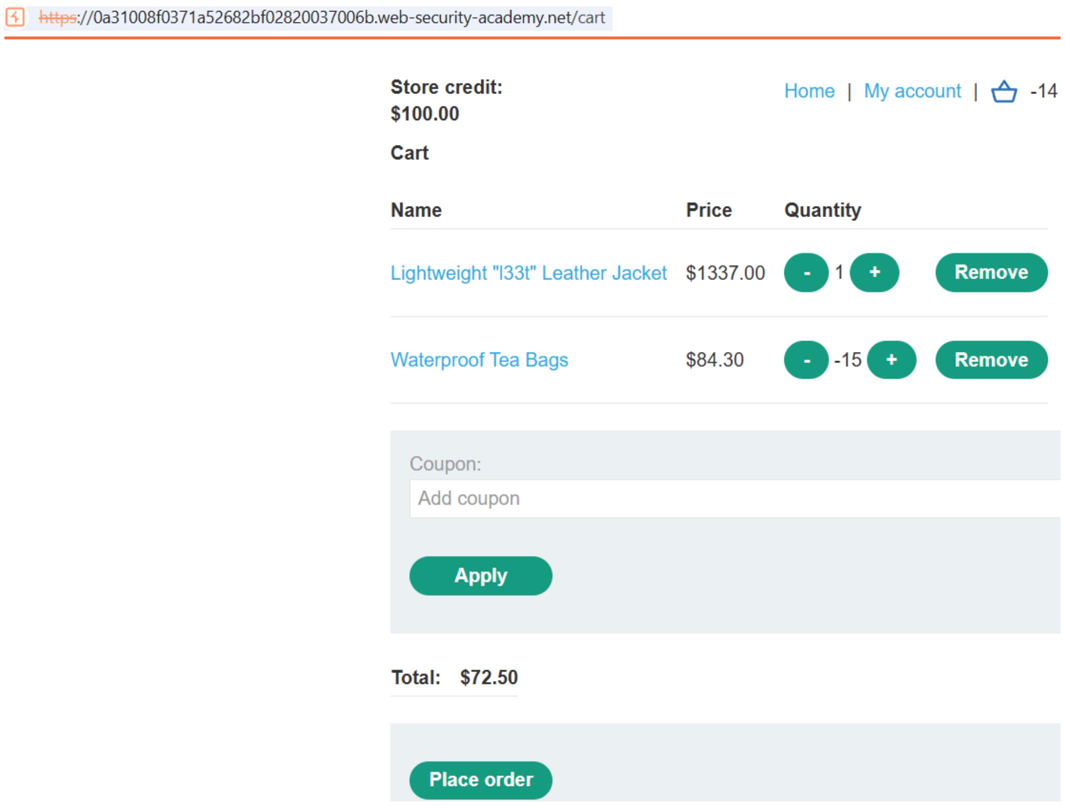
```
 1 POST /cart HTTP/2
 2 Host: 0a31008f0371a52682bf02820037006b.web-security-academy.net
 3 Cookie: [redacted]
 4 Content-Length: 36
 5 Cache-Control: [redacted]
 6 Sec-Ch-Ua: [redacted]
 7 Sec-Ch-Ua-Mobile: [redacted]
 8 Sec-Ch-Ua-Platform: [redacted]
 9 Accept-Language: [redacted]
10 Origin: https://0a31008f0371a52682bf02820037006b.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: [redacted]
13 User-Agent: [redacted]
14 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.
   8,application/signed-exchange;v=b3;q=0.7
15 Sec-Fetch-Site: [redacted]
16 Sec-Fetch-Mode: [redacted]
17 Sec-Fetch-User: [redacted]
18 Sec-Fetch-Dest: [redacted]
19 Referer: https://0a31008f0371a52682bf02820037006b.web-security-academy.net/product?productId=4
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 productId=4&redir=PRODUCT&quantity=-15
```

Image 12: Business logic vulnerability, and POST request with quantity=-15

```
POST /cart HTTP/2
Host: 0a31008f0371a52682bf02820037006b.web-security-academy.net
Cookie: [...]
Content-Length: 36
Origin: https://0a31008f0371a52682bf02820037006b.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: https://0a31008f0371a52682bf02820037006b.web-security-
academy.net/product?productId=4
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
productId=4&redir=PRODUCT&quantity=-15
```

Example 3: POST request with item quantity=-15, code example

The image below shows the cart when the second item has been added. The cart now contains one “Lightweight l33t leather jacket”, worth $1337.00, and minus 15 “Waterproof Tea Bags”, worth $84.30. Since the “Waterproof Tea Bags” quantity is minus 15, their real value is $-1264.50, which makes the total amount to pay $72.50.



~~https~~://0a31008f0371a52682bf02820037006b.web-security-academy.net/cart

Store credit:
$100.00

Home | My account | -14

Cart

Name	Price	Quantity	
Lightweight "l33t" Leather Jacket	$1337.00	- 1 +	Remove
Waterproof Tea Bags	$84.30	- -15 +	Remove

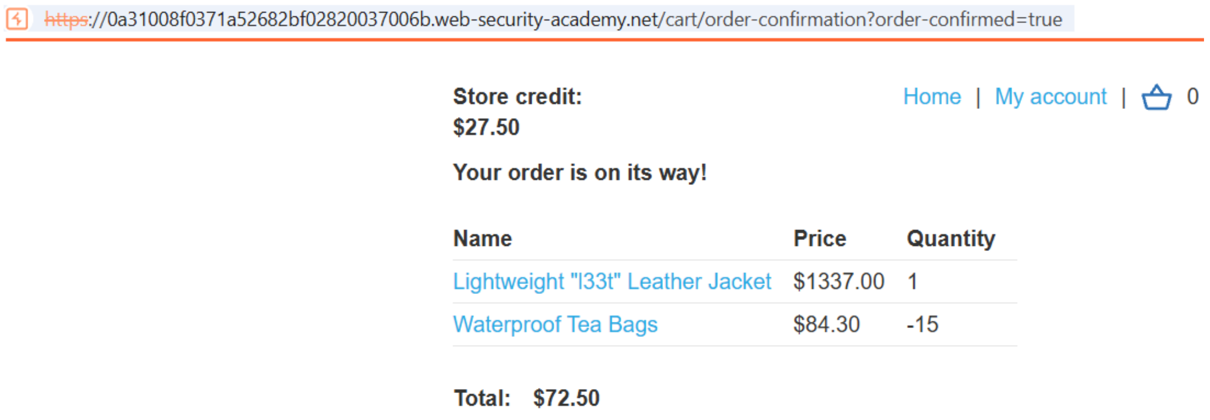
Coupon:
Add coupon
Apply

Total: $72.50

Place order

Image 13: Cart containing both items

The image below shows confirmation that the order has been placed and the total amount that has been paid is $72.50.



~~https~~://0a31008f0371a52682bf02820037006b.web-security-academy.net/cart/order-confirmation?order-confirmed=true

Store credit:
$27.50

Home | My account | 0

Your order is on its way!

Name	Price	Quantity
Lightweight "l33t" Leather Jacket	$1337.00	1
Waterproof Tea Bags	$84.30	-15

Total: $72.50

Image 14: Order confirmation

Recommendations

It is recommended to avoid making implicit assumptions about parts of the application and user behaviour. Identify what early assumptions that has been made about the server-side state and implement the necessary logic to mitigate these assumptions. It is also recommended to maintain clear design documents in the future, which makes it possible to go back and see what assumptions has been made at each stage. Comprehensive and thorough documentation and testing of the application is highly necessary and recommended to mitigate broken business logic.

For more information on mitigating broken Business Logic, see:

Open Web Application Security Project – Business logic vulnerability[3] – Test Business Logic Data Validation[4]

[3] https://owasp.org/www-community/vulnerabilities/Business_logic_vulnerability
[4] https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/01-Test_Business_Logic_Data_Validation