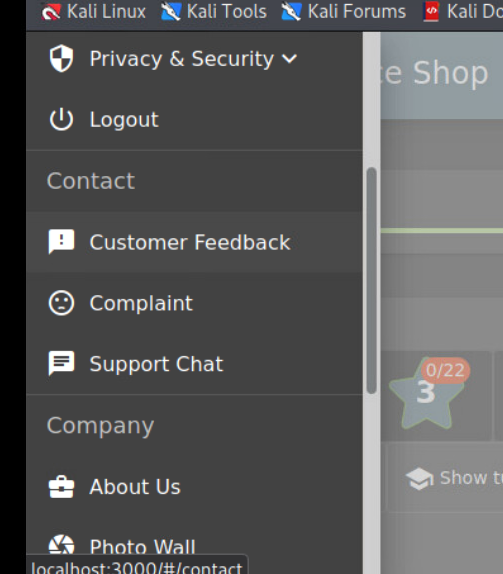
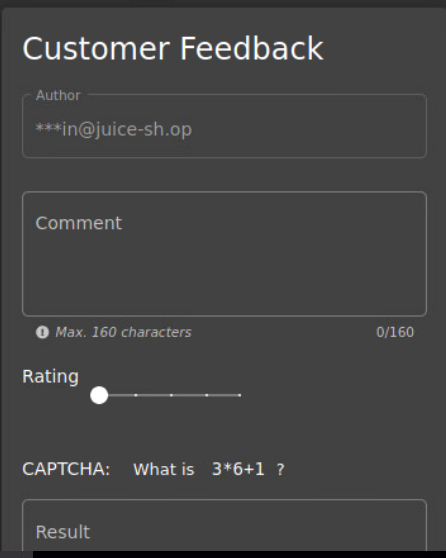


Zero Star Challenge

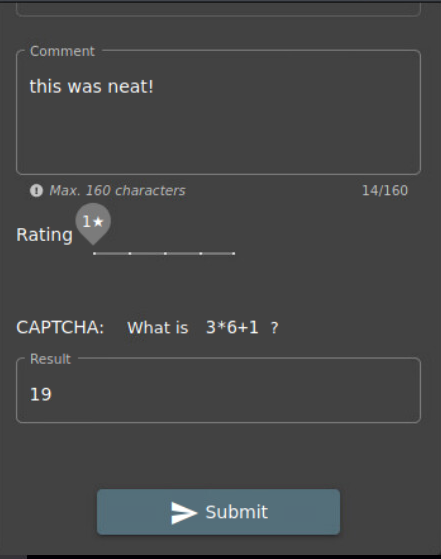
In this exercise we wanted to see if there are any vulnerabilities in the Customer Feedback section. The first steps was to go to the Menu section and select the Customer Feedback section. When selecting this section, a pop up window appears where one can make a customer comment and rating. There are appears to be some checks to insure that a rating is done correctly so we move on to examine the pop-up rating.



Once we are at the Customer Feedback pop up window it shows who the person who will give the feedback is. In the first test we use the administrator login which was discovered in a previous vulnerability and check the common and CAPTCHA boxes to insure that one has to fill them in before leaving a rating.



In the next step we filled in the Comment and CAPTCHA boxes and see that the Submit button is not enabled until a Rating is set. In this case the lowest rating available was a one-star rating. Now we want to find out if we can leave a lower rating, which is an unwanted rating of zero stars.



We try to make a rating of zero but are unable unless all three settings are set correctly. The next step will be to look at the code regarding the actual Submit button in the Element section of the code to see if it can be altered while in use. For this test we made a user called [dacarbaj@aol.com](mailto:dacarbaj@aol.com) and used it as the login user trying to make the rating.

A screenshot of a computer

Description automatically generated

So we fill in the comment and the CAPTCHA boxes and prepare to look at the Submit button code.

A screenshot of a computer

Description automatically generated

The next step we right clicked on the Submit button and it opened the Dev Ops code for the submit button. In there we changed the ‘primary-mat.button-disabled’ to ‘primary-mat.button-enabled’ and then we deleted the disable = ‘true’ code and as soon as we did this, a zero star comment was able to be sent through the code. The following diagram shows the Zero Stars Rating went through.

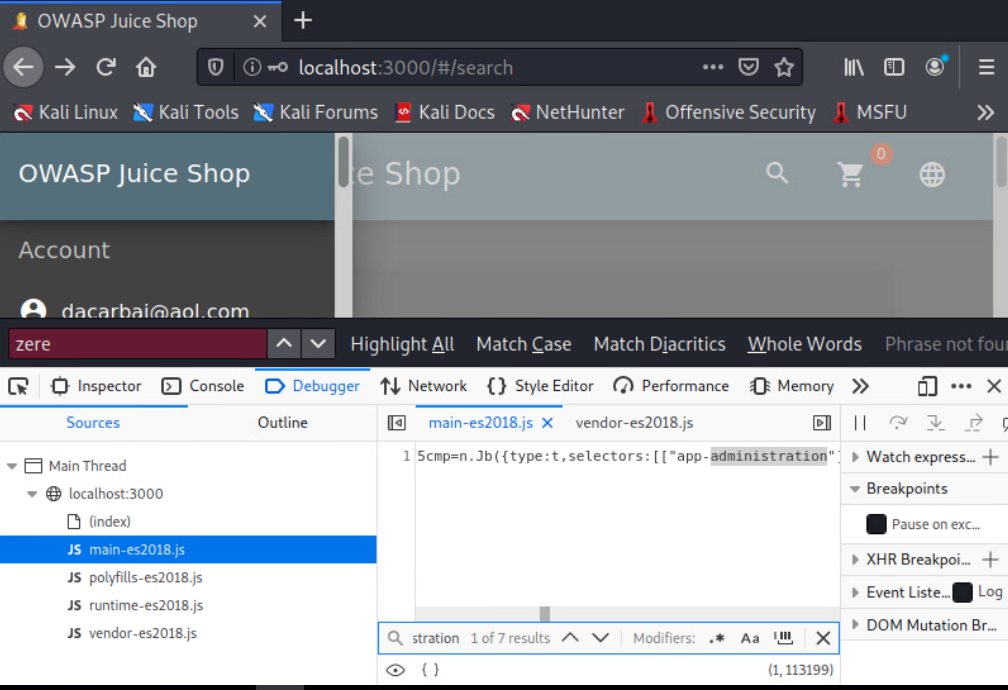
A screenshot of a computer

Description automatically generated

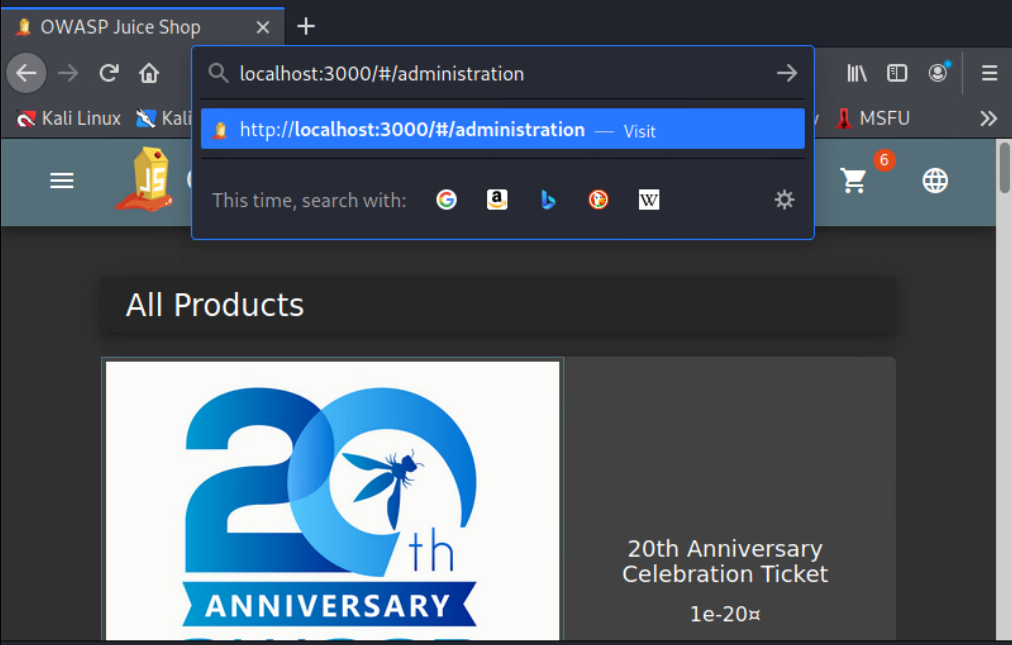
This is an improper Input Validation vulnerability. [ASVS V5](https://wiki.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding), [API6:2019](https://owasp.org/www-project-api-security)

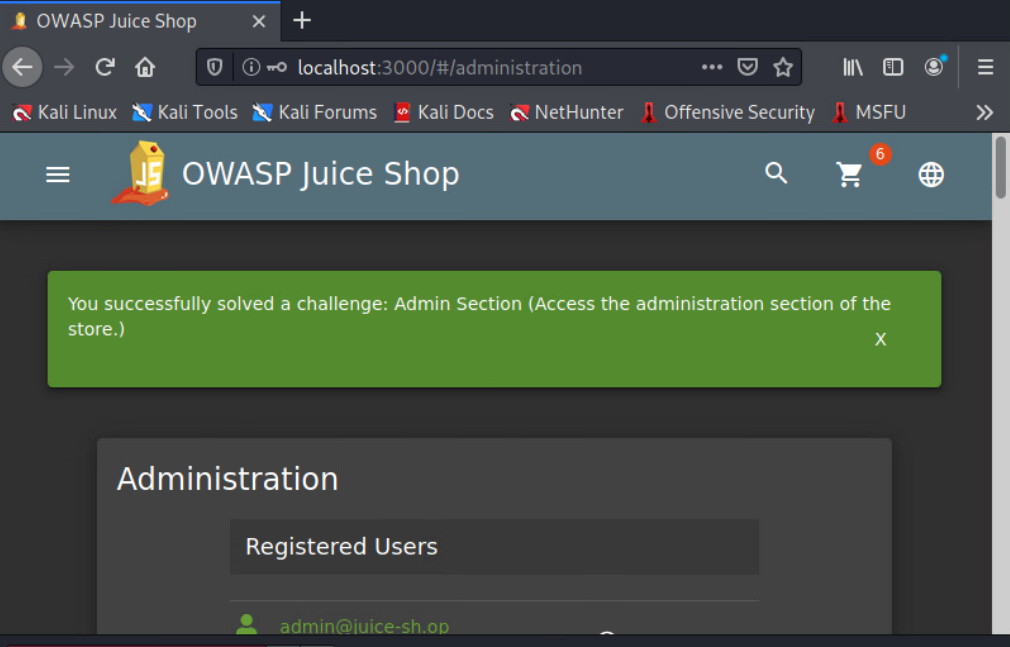
To mitigate this vulnerability Input Validation needs to be repeated on the server side before it ever reaches a database. Just because a form is set up to validate inputs, that doesn’t mean the server will reject information in modified packets.

In the next step we wanted to see if we can reach the administration section of the code. We used a similar method that was done to find the scoreboard page in the previous inspection. We opened Dev ops and looked for the administration name on the page. We used administrator, admin, administrator, and found that admin matched administration. We then used the [admin@juice-sh.op](mailto:admin@juice-sh.op) and the password admin123 to login into the application.

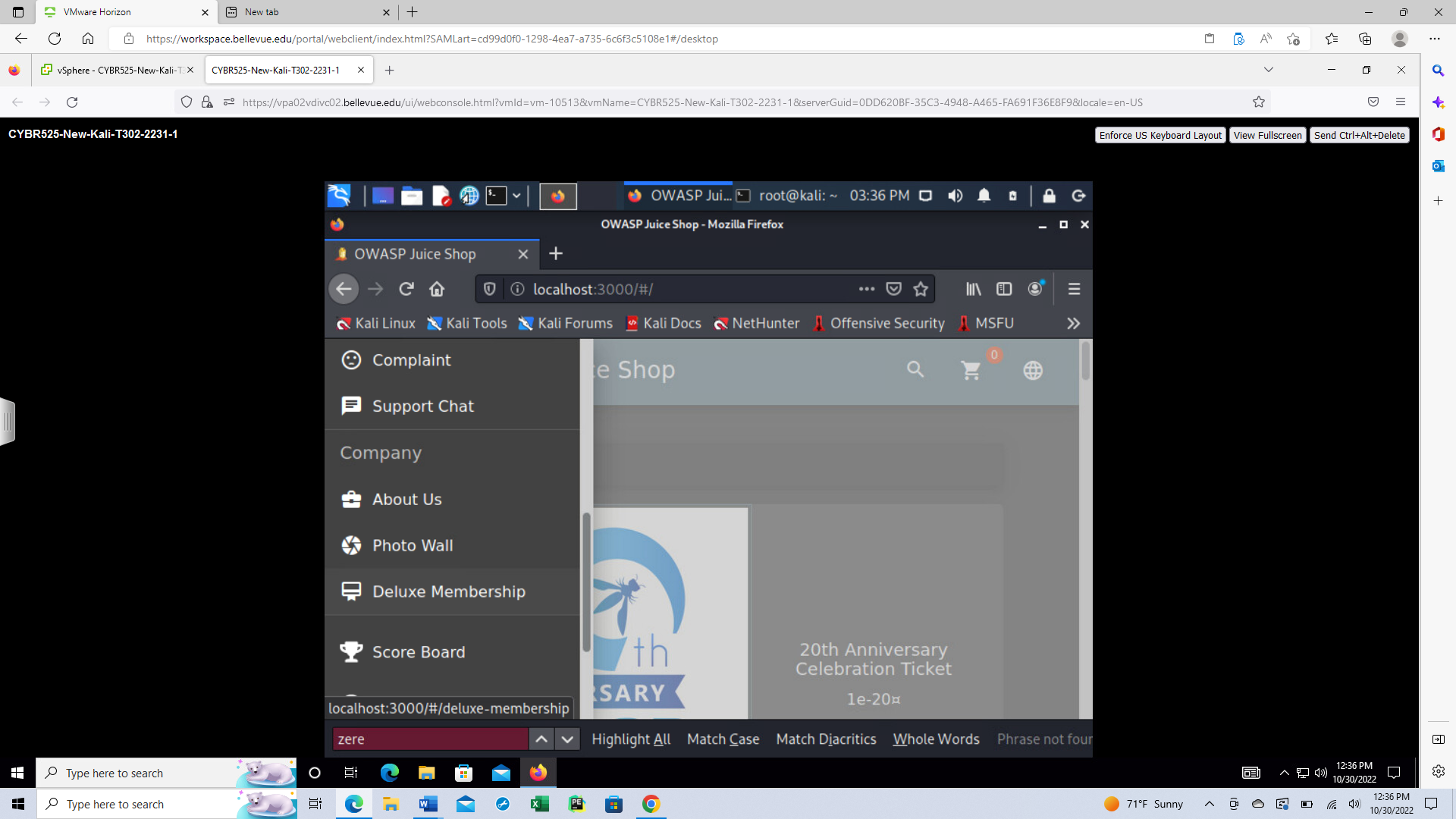


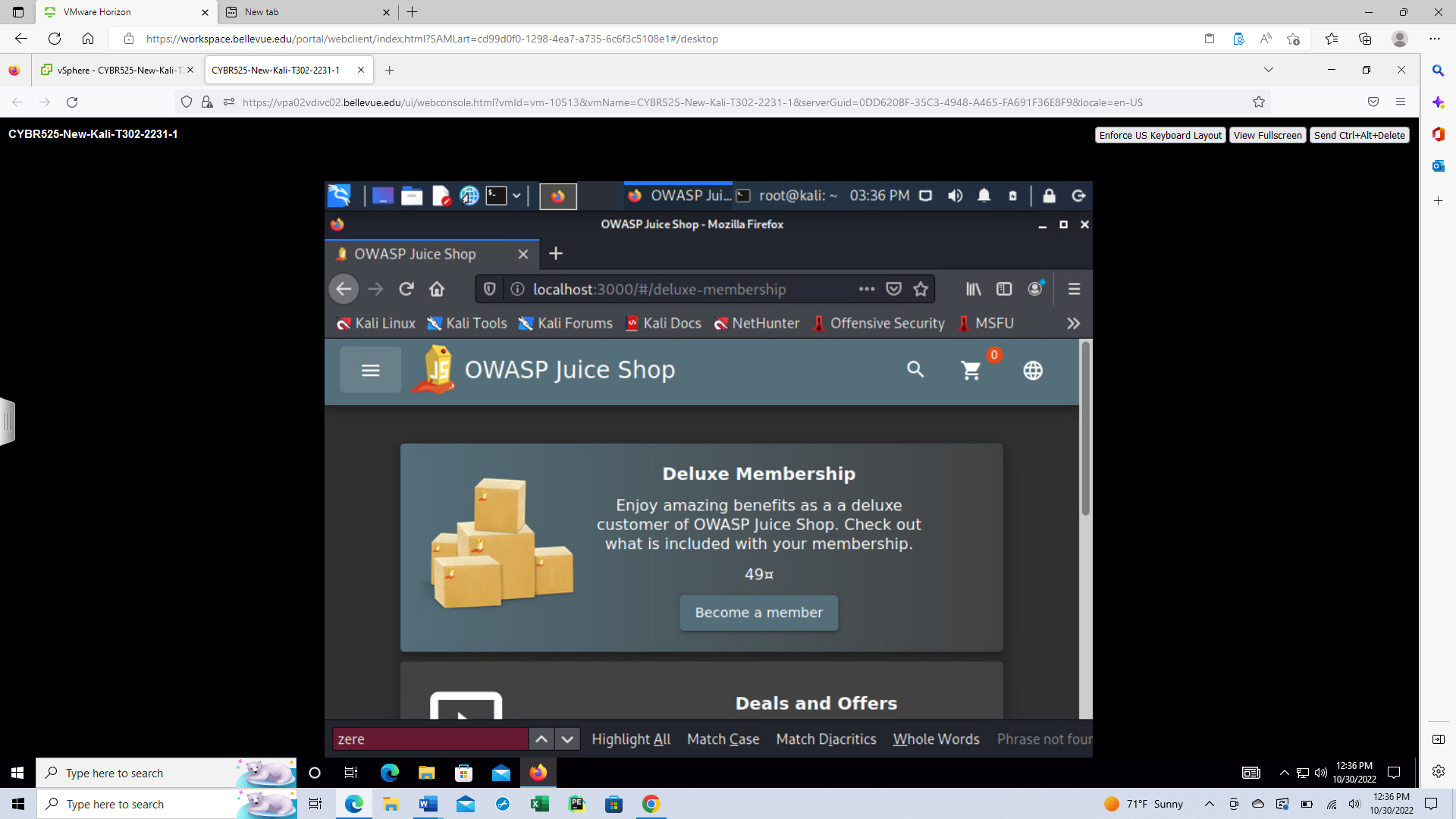
Then we typed in <http://localhost:3000/#/administration> and it landed on the administration page. The figure below shows the login followed by the figure showing that he Juice Shop can be vulnerable to having its administration page vulnerable to an attack.

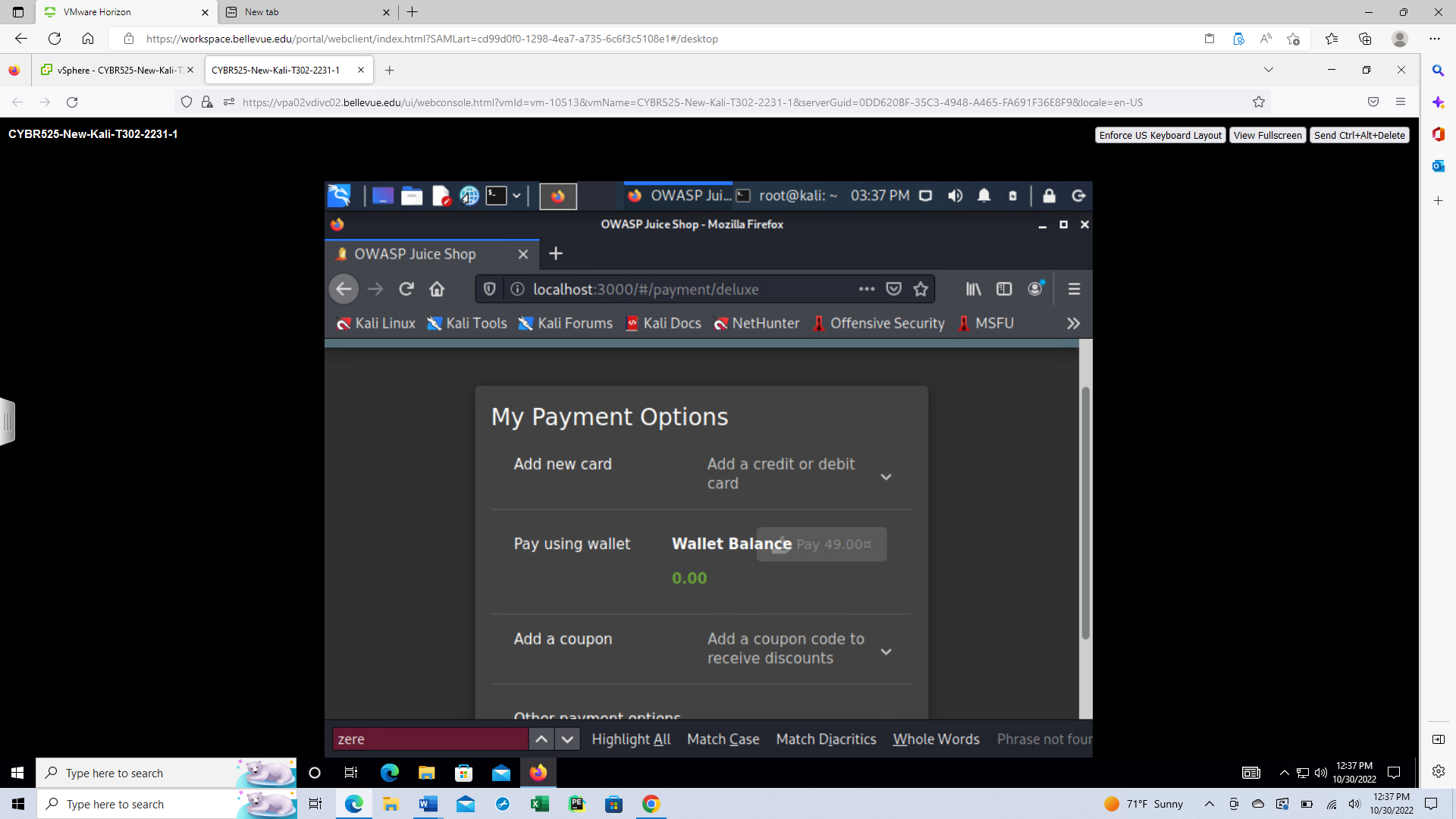


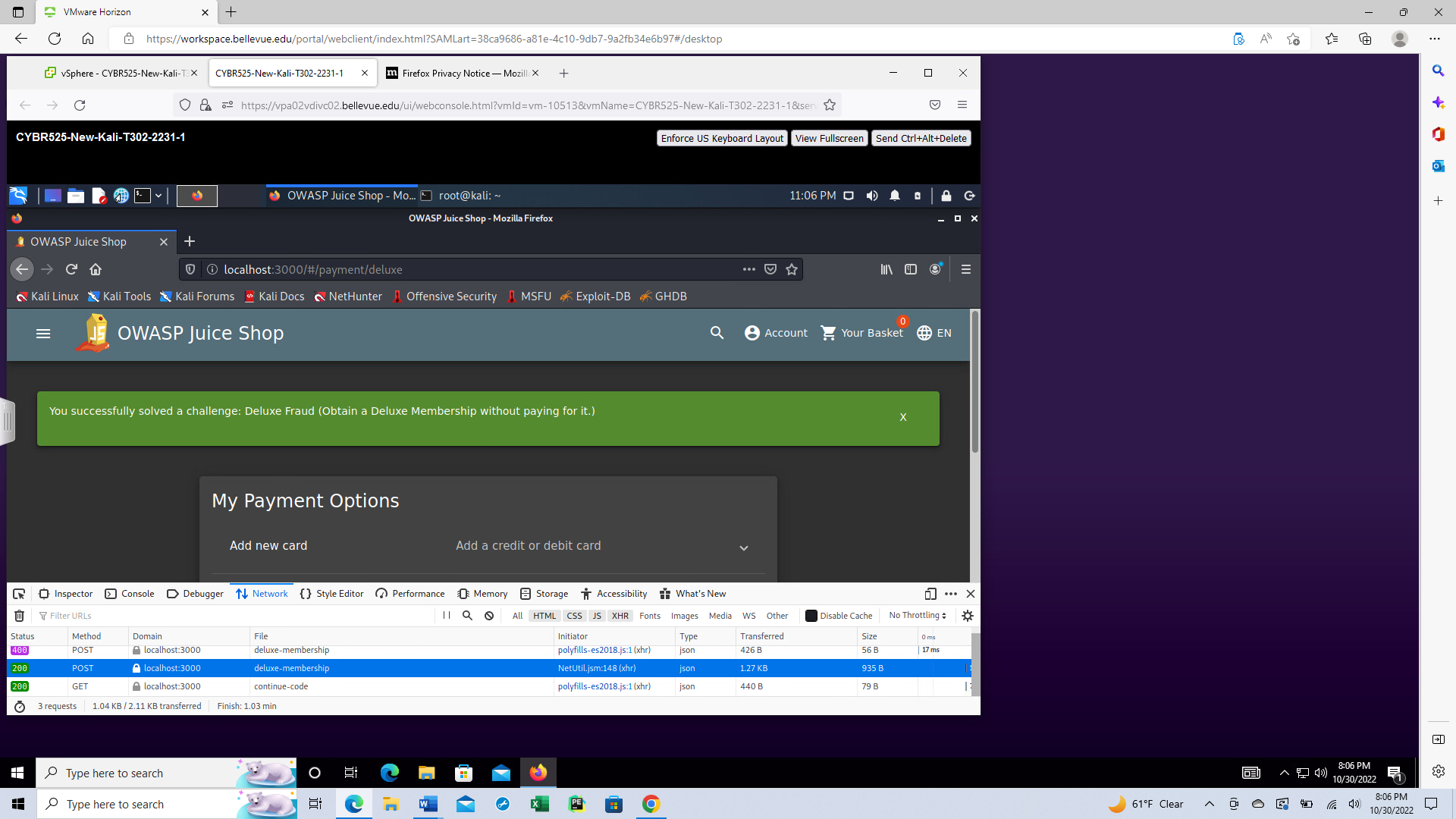


Deluxe membership









This a Broken Authentication vulnerability [A7:2021](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/), [API2:2019](https://owasp.org/www-project-api-security), [P6:2021](https://owasp.org/www-project-top-10-privacy-risks) . The Juice Shop needs to ensure that administrator login is not easily authenticated by an attacker. This will ensure that one cannot easily get to the Administration page.

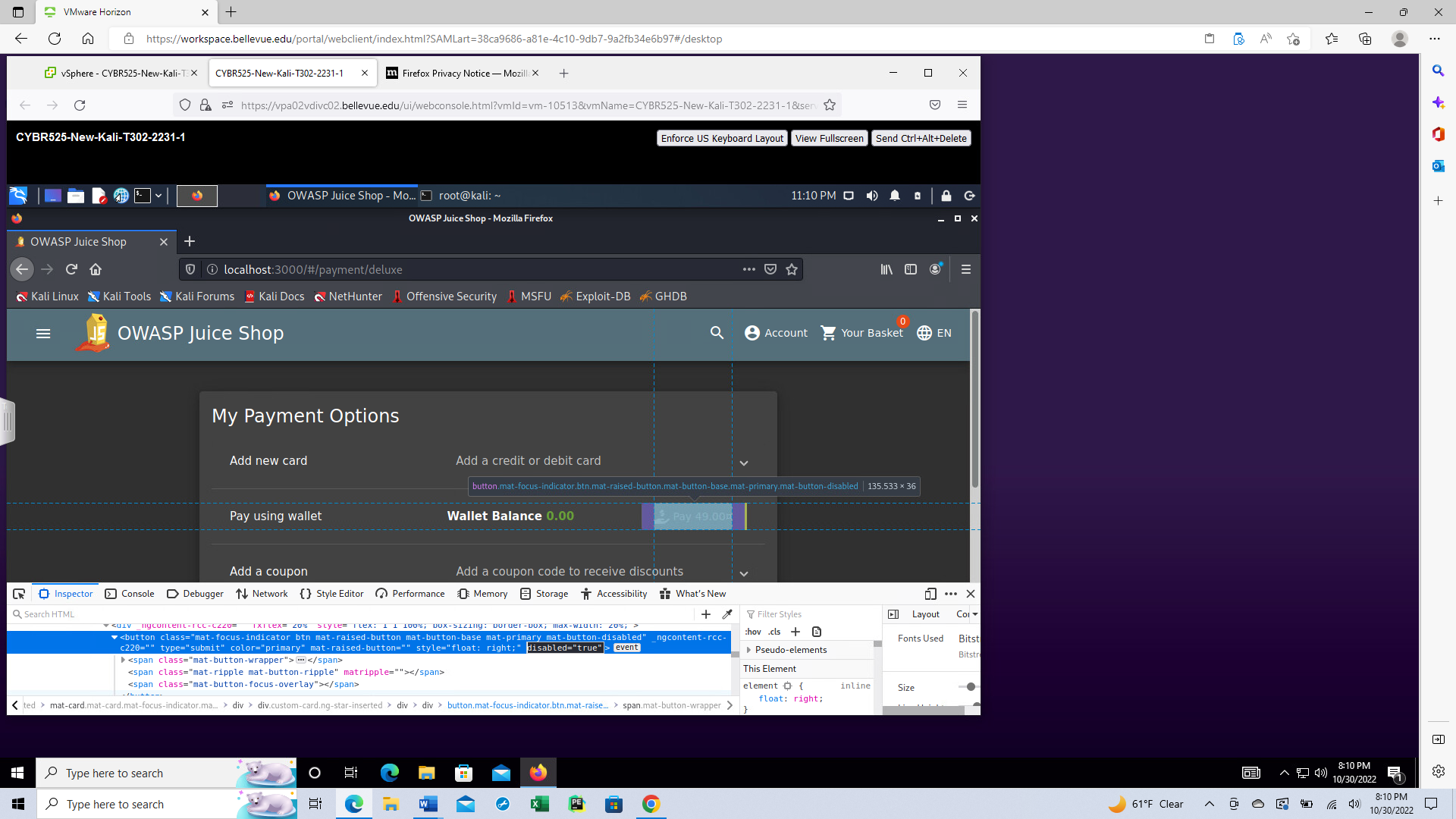
\

In the final test we want to see if we can make a payment without actual money. Thus, we try to attack the payment by wallet without adding any credit card or funds to the wallet. Here we go the wallet and select the Pay button but it is not active. So we Inspect the Element using DevOPs

A screenshot of a computer

Description automatically generated

First we use the previous trick of Invalidation by activating the Pay button by deleting the delete = ‘true’ and setting the ‘primary-mat.button-disabled’ to ‘primary-mat.button-enabled’ and we press the button. But we find even though the button is active nothing happens.

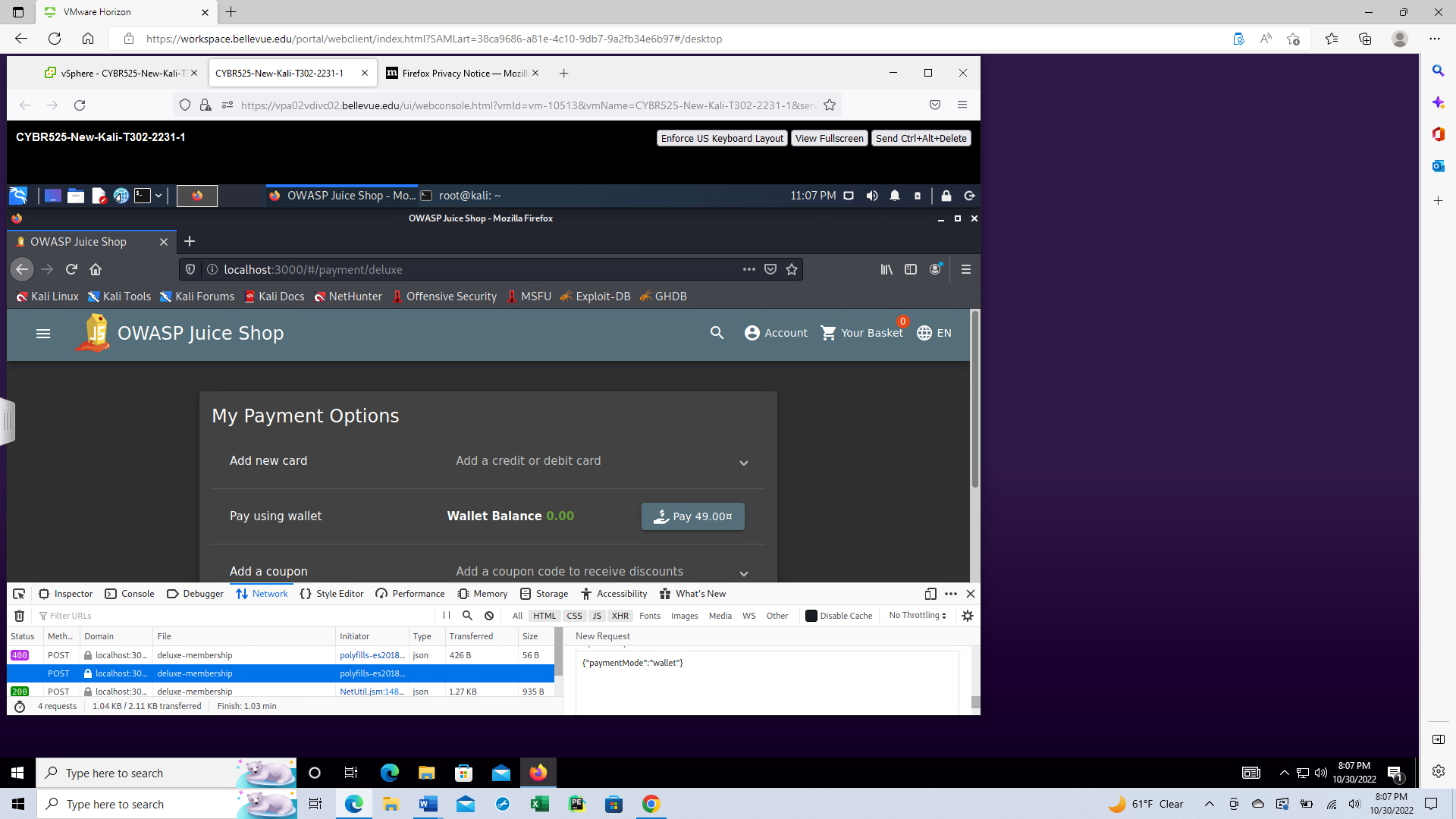


So then we use DevOps to go into Network Mode and see that when the button is pressed. It sends and error message.

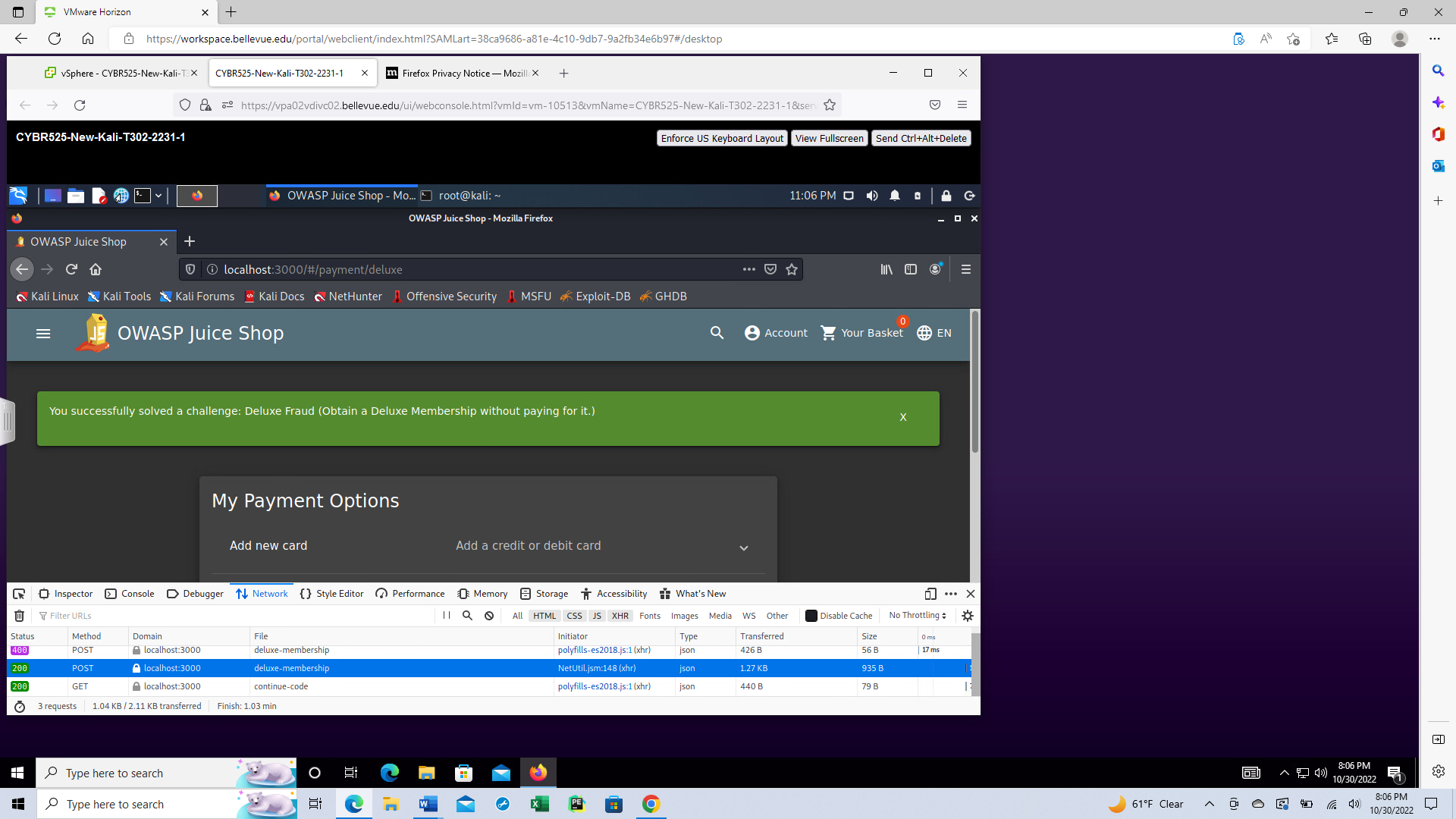
A screenshot of a computer

Description automatically generated

Scrolling down on the message we see paymentMode = “Wallet”. We do a Edit and Resend and this is changed to “none” for payment mode.



And we find that we can now use the wallet to purchase items without using any funds.



This is also an improper Input Validation vulnerability. [ASVS V5](https://wiki.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding), [API6:2019](https://owasp.org/www-project-api-security)

To mitigate this vulnerability Input Validation needs to be repeated on the server side before it ever reaches a database. This is especially important as we don’t want an attacker to be able to order items illegally and not pay for them.