Exploiting Parameter Tampering

Abstract

The objective of this project was to analyse and exploit vulnerabilities in the OWASP Juice Shop web application, focusing on flaws introduced by Parameter Tampering and weak Business Logic. Five challenges were successfully exploited: View Basket (IDOR), Deluxe Fraud (Price Manipulation), Forged Feedback (Unauthorized Action), Payback Time (Transaction Logic Flaw) and Product Tampering (Unauthorized Data Update). The final output confirms that the application critically fails to implement server-side validation and proper authorization across multiple API endpoints, leading to severe financial, privacy, and integrity risks.

Target System Details

Parameter	**Description**
Site Name	OWASP Juice Shop
Project URL	https://owasp.org/www-project-juiceshop/
Category/Type	Intentionally Insecure Web Application (E-Commerce Simulation)
Primary Usage	Security training and penetration testing practice.

Technology Stack & Testing Environment

Component	**Technology Used**
JavaScript Frameworks	Angular, Zone.js
Database	MySQL
Programming Languages	TypeScript, Node.js, JavaScript, PHP
Web Servers	Apache HTTP Server, Nginx
Testing Environment	Docker Container on Ubuntu Live Server

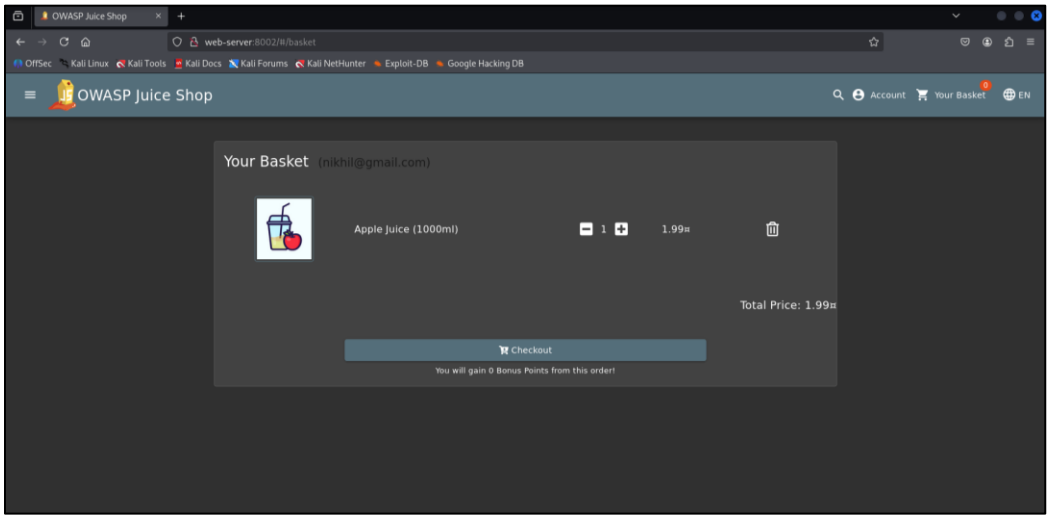
1. View Basket Challenge

The web application allows users to view the contents of their shopping basket. However, upon examining the network traffic with Burp Suite, it was discovered that the request to fetch the basket details includes a user-specific identifier (e.g., basket ID) which is vulnerable to manipulation.

Proof of Concept

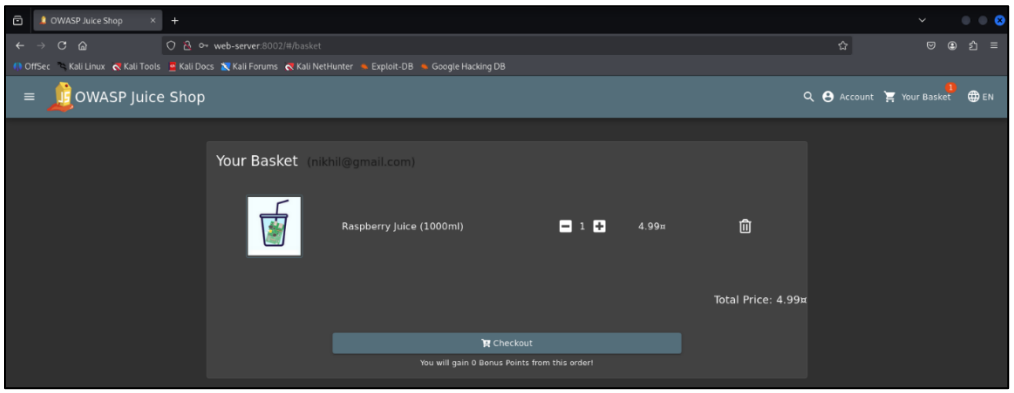
Intercept the HTTP Request:

- Using Burp Suite, the HTTP request to retrieve the user's own basket was intercepted.
- The request URL is: http://web-server:8002/rest/basket/6 , where 6 is the basket ID for the current user.
- We obtain our basket.



Modify the Basket ID:

- Altered the basket ID in the intercepted request from 6 to 3, attempting to access another user's basket details.
- View Results:
- The modified request was forwarded, and the response contained the details of another user's basket, confirming the presence of broken access control.



Remediation

To mitigate such vulnerabilities in real applications, the following steps should be considered:

- **User Session Validation:** Ensure that each request to sensitive information like a shopping basket is validated against the user's session to confirm they are authorized to view only their data.
- **Use of Robust Access Control Mechanisms:** Implement role-based access control (RBAC) or attribute-based access control (ABAC) to enforce strict permissions on data access.
- **Regular Security Audits:** Conduct regular security audits and penetration testing to identify and fix access control issues before they can be exploited.

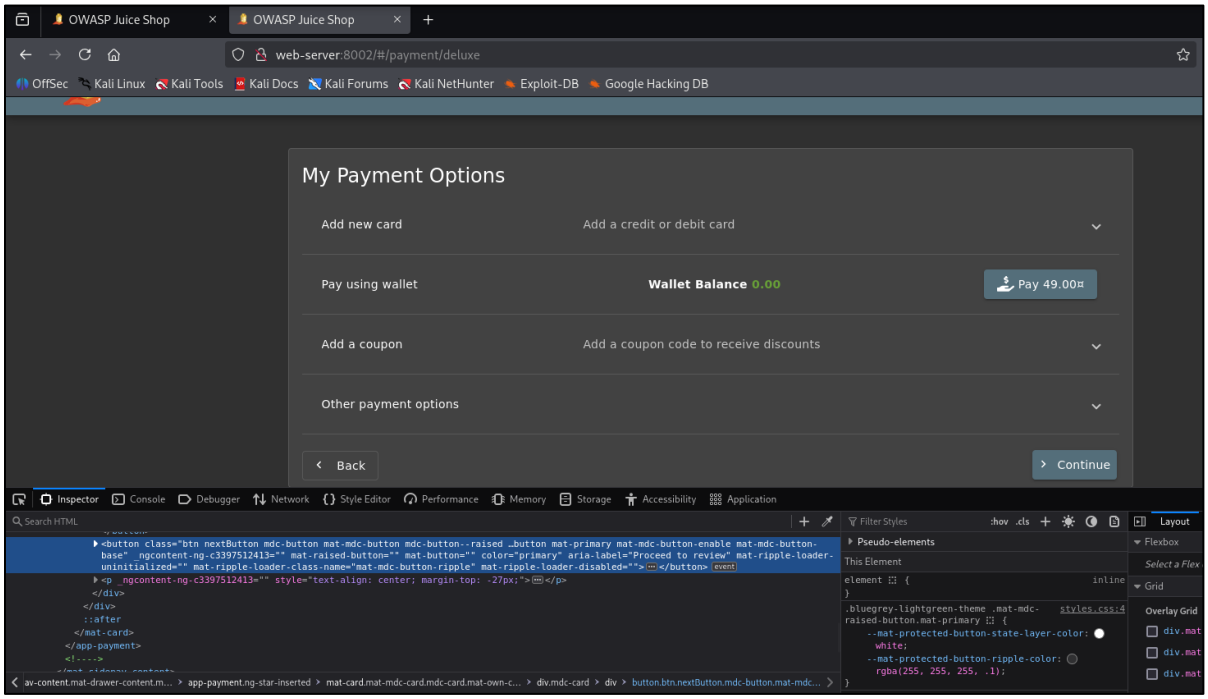
2. Deluxe Fraud Challenge

The “Deluxe Fraud” challenge involves manipulating the web application to obtain a Deluxe Membership without proper payment. This challenge tests skills in identifying and exploiting vulnerabilities related to payment and authorization mechanisms.

Proof of Concept

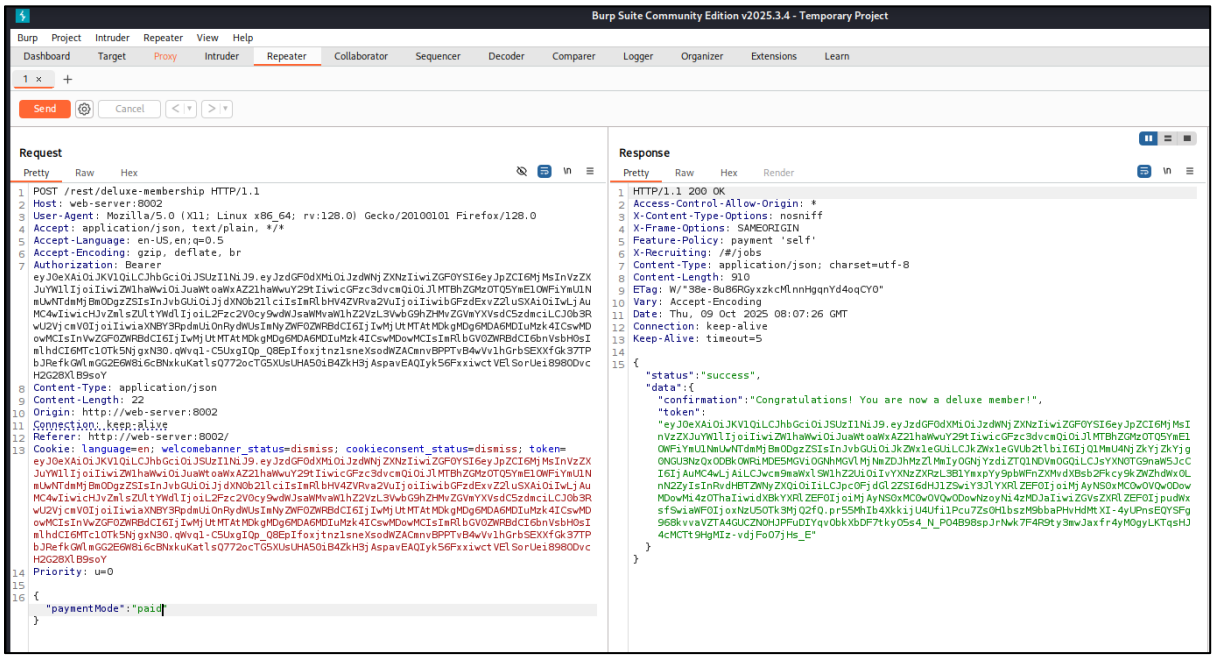
Exploring Payment Gateway:

- Proceeded to the payment page from deluxe membership page http://web-server:8002/#/payment/deluxe .
- Open devtools and inspect the pay and continue buttons.
- Remove the disabled=“true” attribute from the element to enable it and click on the button to initiate payment.



Intercept the HTTP Request:

- Using Burp Suite, intercept the HTTP request of payment.
- See that there is a POST request sent, which only contains one parameter in the request payload, “paymentMode”, which is set to “wallet”.
- Change the paymentMode parameter to an empty string or “paid” and press send.



Remediation

To prevent such vulnerabilities in real-world applications:

- **Enhance Input Validation:** Ensure that all inputs, especially those related to payment operations, are rigorously validated both on the client-side and server-side.
- **Secure Payment Logic:** Implement robust checks on the server-side to verify that payment details are correct and complete before processing transactions.
- **Use Secure Payment Gateways:** Integrate with reputable payment gateways that provide additional security checks and validations.

3. Forged Feedback Challenge

The “Forged Feedback” challenge involves exploiting insufficient access controls to post feedback under another user's name, illustrating vulnerabilities related to user identity management within a web application.

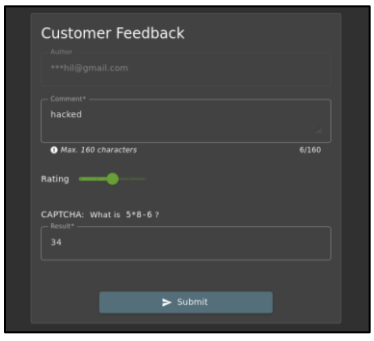
Proof of Concept

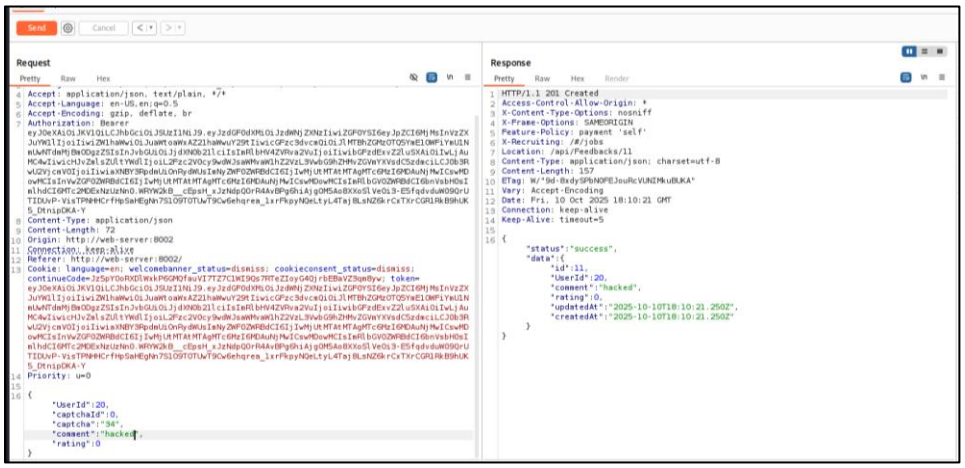
Intercept the Request:

- Using Burp Suite, capture the HTTP request sent when feedback is submitted.

Manipulate the Parameters:

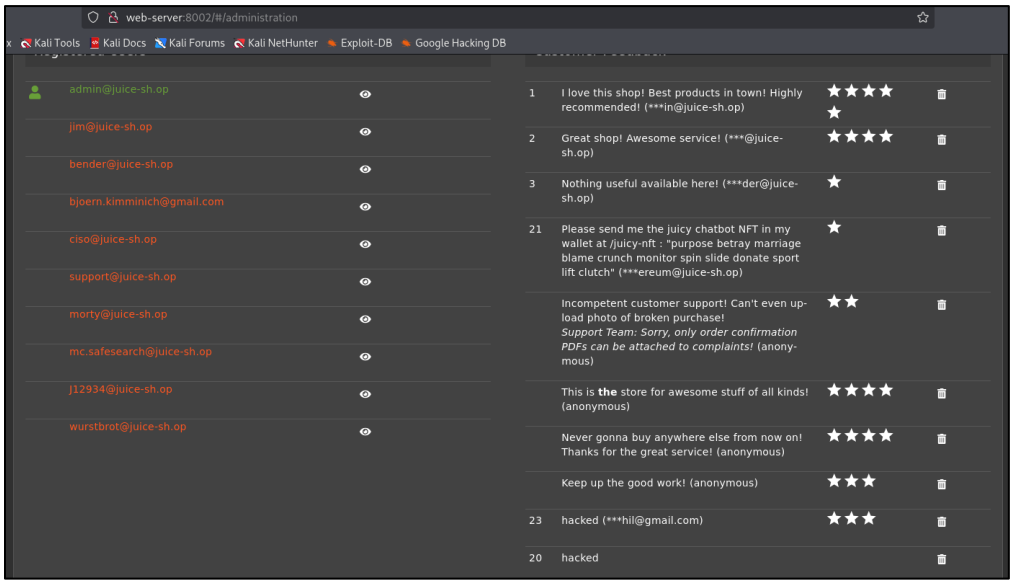
- Modify the parameters like UserId in the intercepted request.
- Resend the modified request to see if the feedback gets posted with the changes.





Verify the Outcome:

- Check the application to confirm whether the feedback appears under the other user's profile or feedback history.
- We can check the feedbacks of the users in administration section.



Remediation

To prevent such vulnerabilities in real-world applications:

- **Enhanced Server-Side Validation:** Ensure that all sensitive actions, such as posting feedback, include server-side checks to confirm that the user associated with the session is the same as the user the action is being performed for.
- **Use Session Management:** Implement secure session management practices that map session IDs to user IDs securely. Actions should be authorized based on session ownership rather than relying on user-provided data like user IDs in the request.
- **Role-Based Access Control (RBAC):** Enforce RBAC to ensure that users can only perform actions that correspond to their roles and permissions.

4. Payback Time Challenge

The “Payback Time” challenge involves exploiting a business logic flaw to achieve unauthorized financial gain through manipulating product pricing in an e-commerce application.

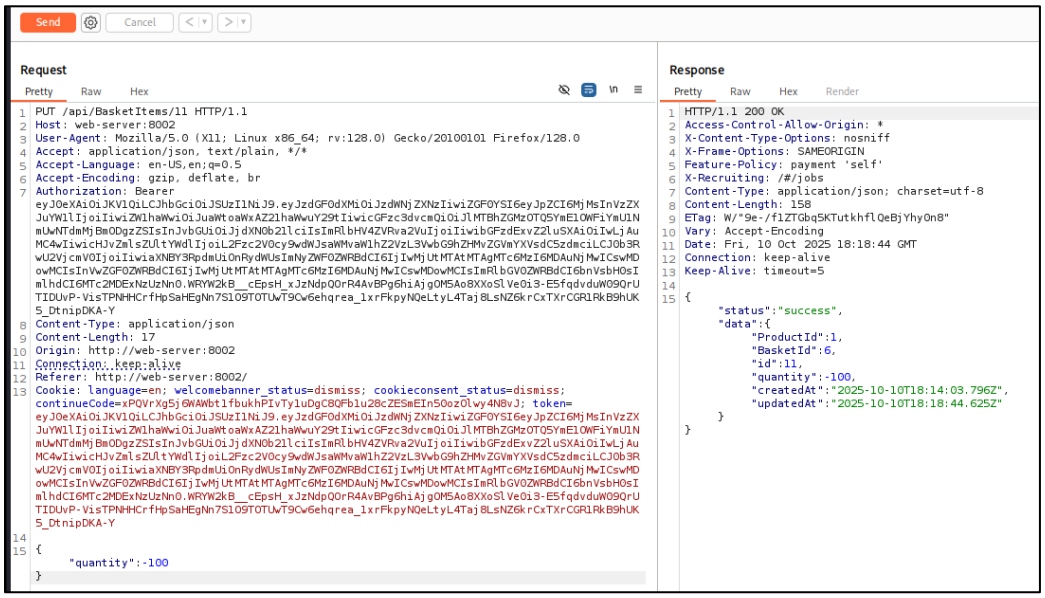
Proof of Concept

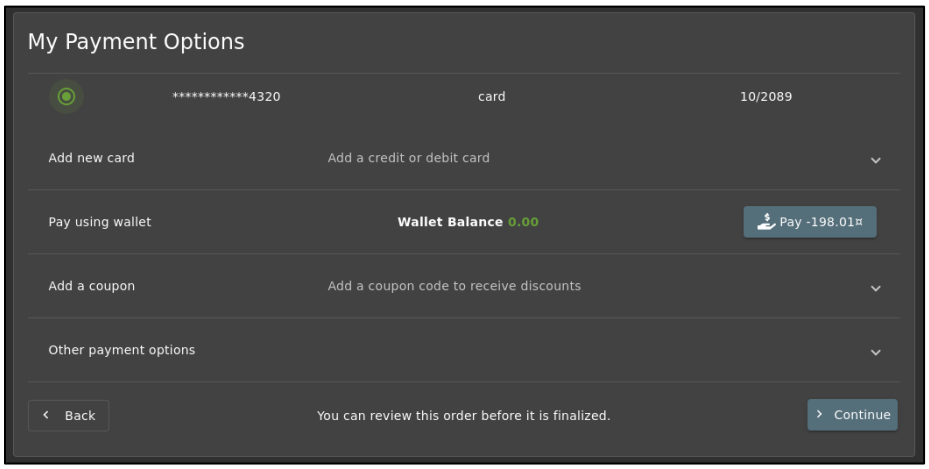
Intercept the Request:

- Using Burp Suite, capture the HTTP request of the user basket, http://web-server:8002/api/BasketItems/11 .

Manipulate Product Pricing:

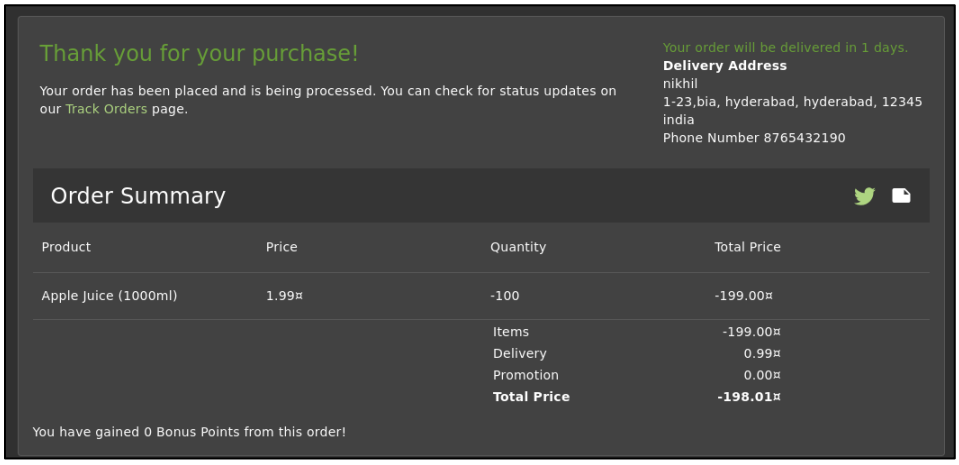
- Modified the intercepted request, setting the price parameter to a negative value.
- The server accepted this request.





Exploit Negative Pricing:

- Purchased the product with the negative price.
- As the system processed the negative amount, it resulted in a credit to the account rather than a charge, effectively leading to a payout.



Remediation

To prevent such vulnerabilities in real-world applications:

- **Proper Input Validation:** Ensure that all inputs, especially those related to financial transactions, are validated both client-side and server-side to prevent manipulation.
- **Secure Payment Logic:** Implement robust checks on the server-side to verify that payment details are correct and complete before processing transactions.

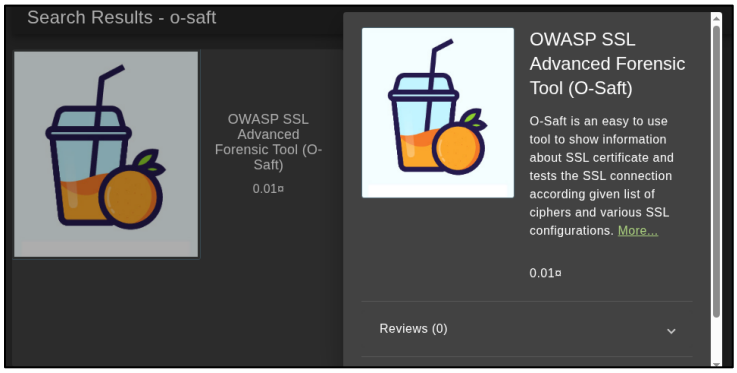
5. Product Tempering Challenge

The challenge, “Product Tempering,” involves changing the href attribute of a hyperlink within the product description of the “OWASP SSL Advanced Forensic Tool (O-Saft)” to point to a new URL: https://owasp.slack.com.

Proof of Concept

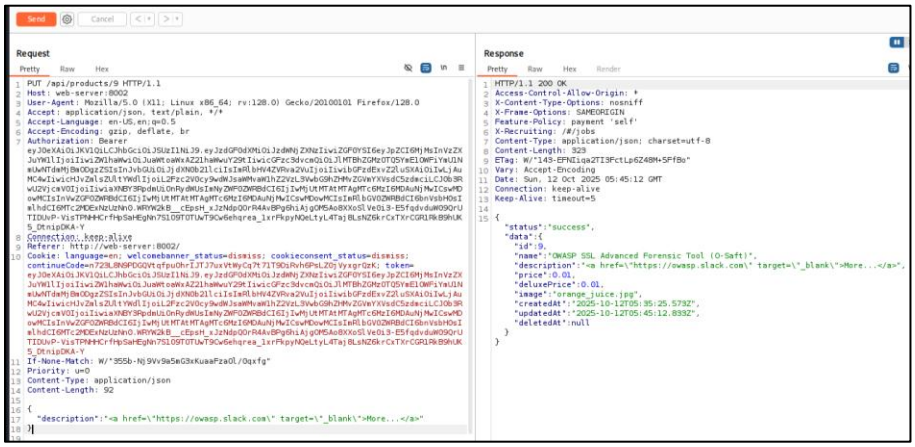
Intercept the Request:

- Using Burp Suite, capture the HTTP request of product “O-Saft”.



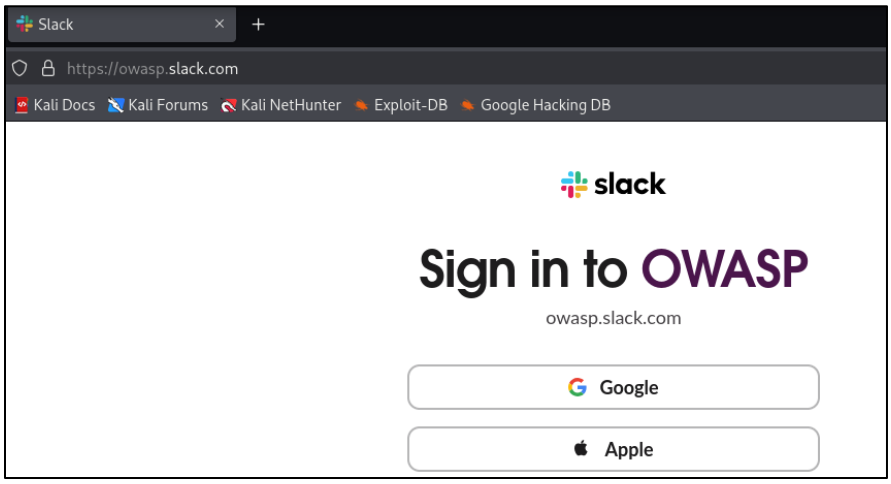
Manipulate the Request:

- Submit a PUT request to http://web-server:8002/api/Products/9 with the {“description”:“ More...”} and application/json as Content-Type.



Verifying the Result:

- Check the product description in the application to ensure that the hyperlink has been changed as intended.
- Confirm that the link points to the new target URL, https://owasp.slack.com.



Remediation

To prevent such vulnerabilities in real-world applications:

- **Implement Strict Role-Based Access Controls (RBAC):** Ensure that only authorized personnel can modify product details.
- **Sanitize and Validate All Inputs:** Properly sanitize and validate all user inputs, especially in product descriptions that allow HTML content.

Conclusion

Parameter Tampering remains a significant threat, successfully exploiting weaknesses in server-side validation and business logic, as demonstrated by the View Basket, Deluxe Fraud, Forged Feedback, Product Tampering, and Payback Time challenges in OWASP Juice Shop. These attacks, ranging from Insecure Direct Object Reference (IDOR) to financial fraud (negative quantities), underscore a critical security failure: relying on client-controlled input for core application logic. Effective mitigation requires adopting a "Never Trust Client Data" approach, ensuring all parameters affecting access control, transactions, and state are strictly validated and managed exclusively by the server.

References

1. OWASP Juice Shop Documentation – https://owasp.org/www-project-juice-shop/
2. Docker Documentation – https://docs.docker.com/
3. OWASP Top 10 Web Security Risks – https://owasp.org/www-project-top-ten/