PT Report

“Simulated Web Application Security Assessment”

Author: Maor Cohen
Time Testing: 18/01/2026 – 21/01/2026
Scoping: N/A
Testing Process: The web application was tested in a controlled environment to identify security weaknesses.
Discovered behaviors were analyzed to determine if they could be exploited to access sensitive information.

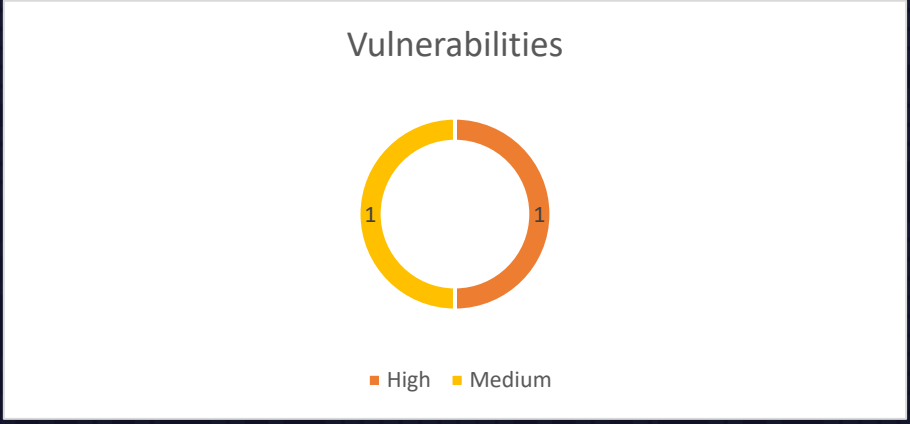
Executive Summary

A security assessment was performed on the web application to evaluate its exposure to misuse and unauthorized access. During testing, multiple weaknesses were identified that allowed sensitive information to be exposed and application functionality to be manipulated in unintended ways due to insufficient server-side controls. If exploited, these issues could lead to data exposure, financial impact, and compromise of the system's integrity.

Conclusions

The security assessment revealed weaknesses that allow both unauthorized access to sensitive internal resources and abuse of application business logic. These issues indicate insufficient server-side validation and improper handling of user-controlled input, which could be exploited to compromise confidentiality and cause financial impact.
The following vulnerabilities were identified and successfully proven (POC):

- SSRF - Server-Side Request Forgery (High)
- Improper Validation of Specified Quantity in Input (Medium)



Exploitation of these vulnerabilities requires minimal technical expertise and may lead to data exposure and financial abuse. Corrective actions are recommended to mitigate these security risks.

Finding Details

VULN-001 Server-Side Request Forgery (High)

Description

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

Reference - https://tinyurl.com/3pwc9ua6

Details

During the assessment, a Server-Side Request Forgery (SSRF) vulnerability was identified within the checkout and receipt generation process. The issue was located in the barcode parameter, which accepted user-supplied SVG content without sufficient validation or restriction on external resource loading.

By modifying the SVG content embedded in the purchase confirmation request, the application was instructed to load a local server resource using a file:// URI. When the purchase was completed, the server processed the malicious input and generated a receipt that rendered the contents of the referenced internal file.

As a result, sensitive server-side data, including an administrative token, was disclosed to the attacker. This action required only a standard user transaction with valid payment information and did not involve elevated privileges. The vulnerability demonstrates insufficient server-side controls over user-controlled input and unrestricted access to internal resources during request processing.

Notes

During reconnaissance, the robots.txt file referenced an administrative JavaScript resource that was publicly accessible but obfuscated. While the file did not directly expose sensitive values during normal access, further analysis revealed references to administrative tokens. Exploitation of the SSRF vulnerability allowed direct access to this file on the server, resulting in disclosure of the administrative token.

Portswigger's Burpsuite tool was used - https://tinyurl.com/4k4hbfv5

Burp Suite is a powerful tool for intercepting and analysing HTTP requests between the client and server. By configuring the browser to use Burp's proxy, all web traffic is captured in the Proxy > HTTP history tab. Captured requests can then be sent to the Repeater tab for detailed inspection.

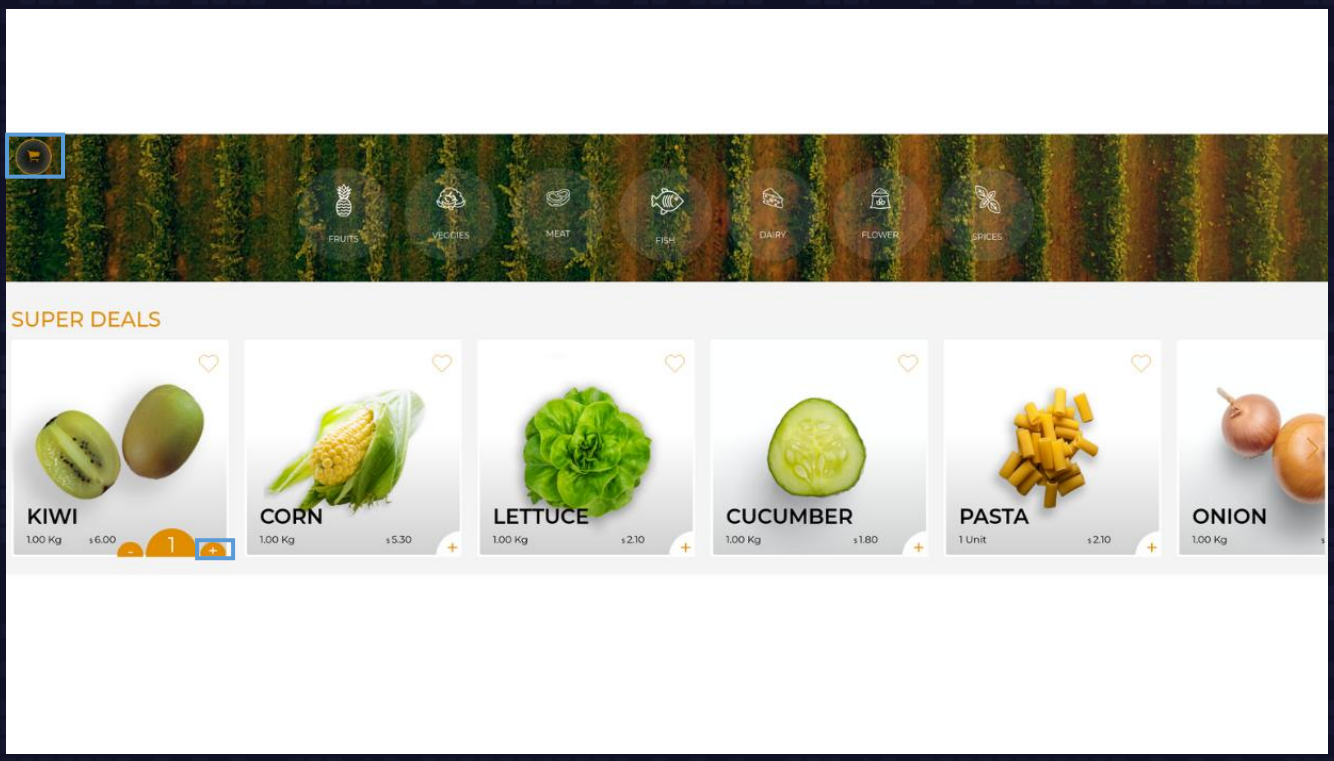


Figure 1: Initial cart interaction prior to exploitation

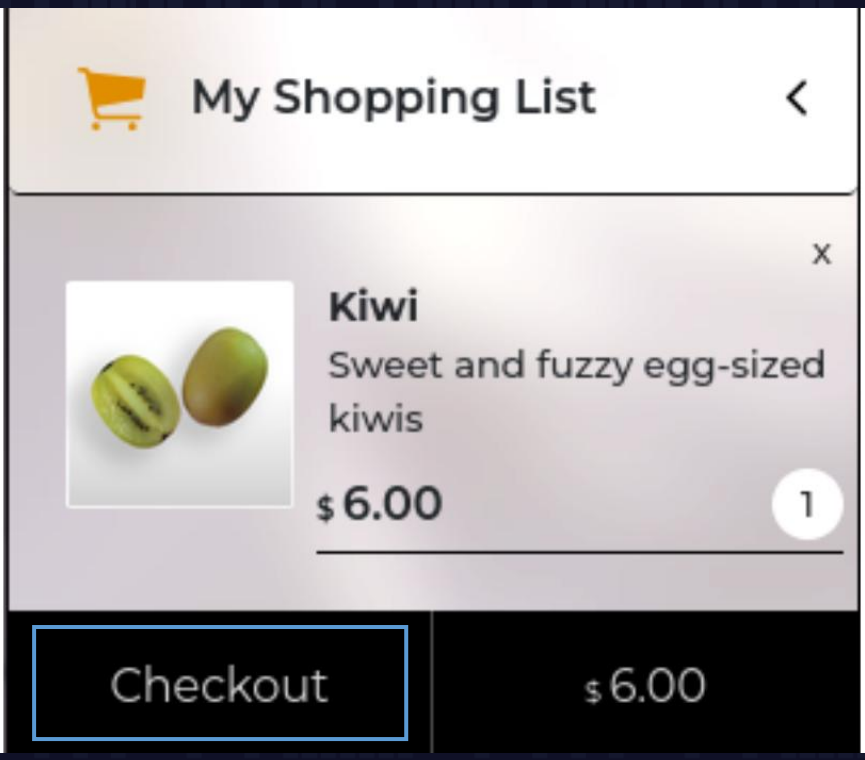


Figure 2: Cart inspection and transition to checkout

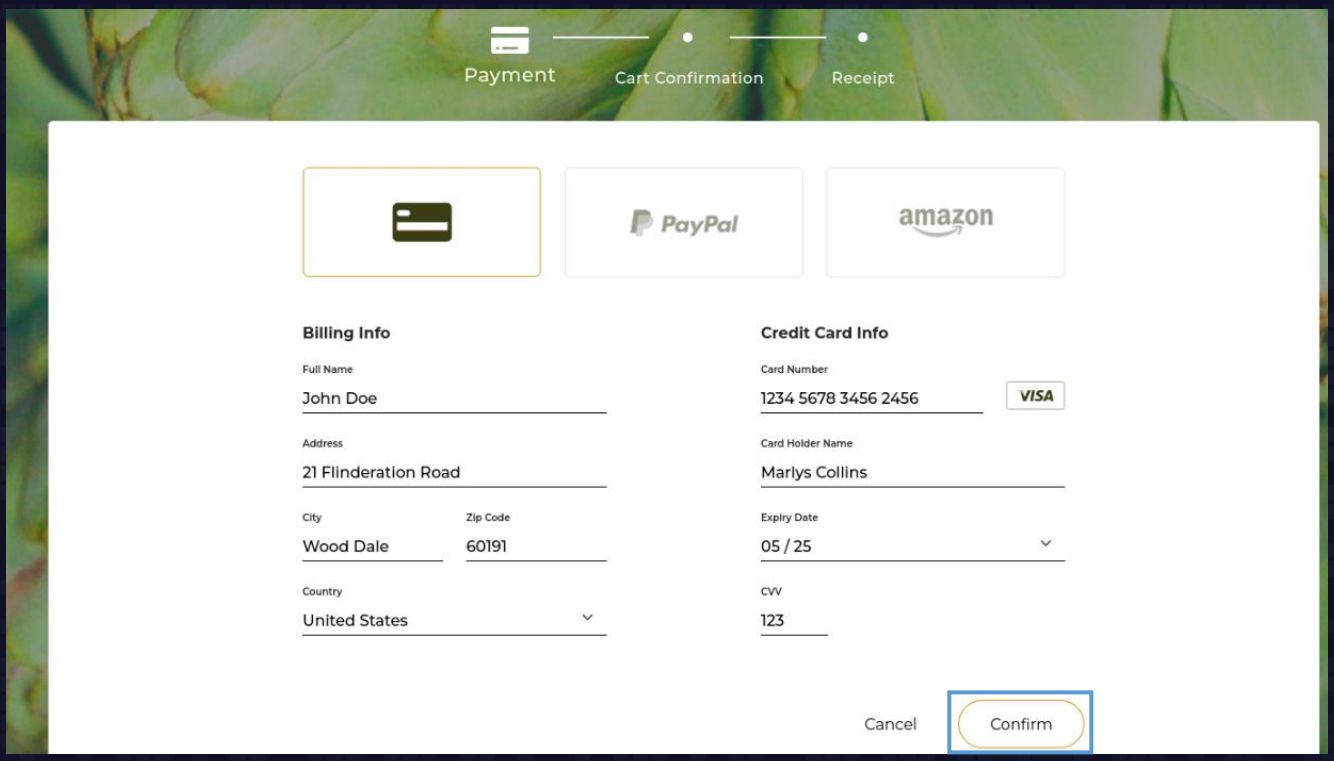


Figure 3: Billing confirmation

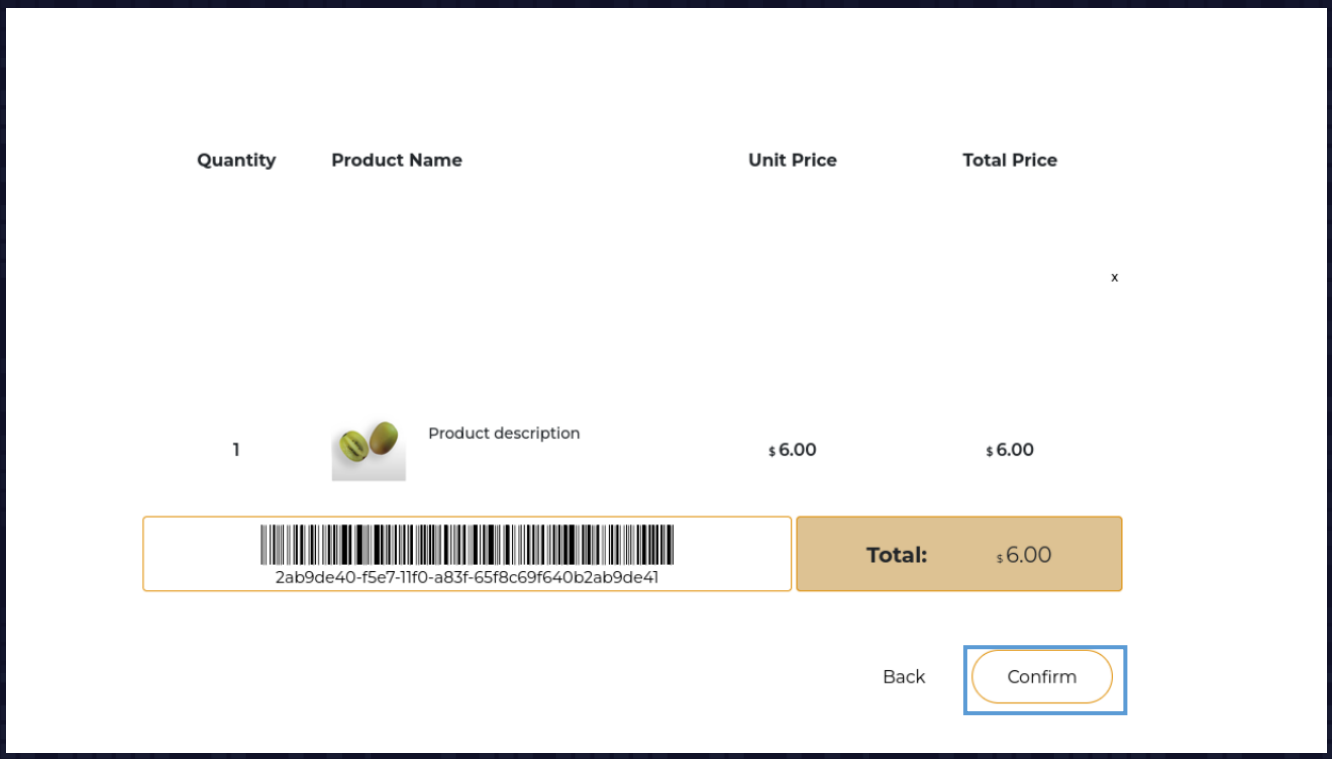


Figure 4: Purchase confirmation which will send the request to the server

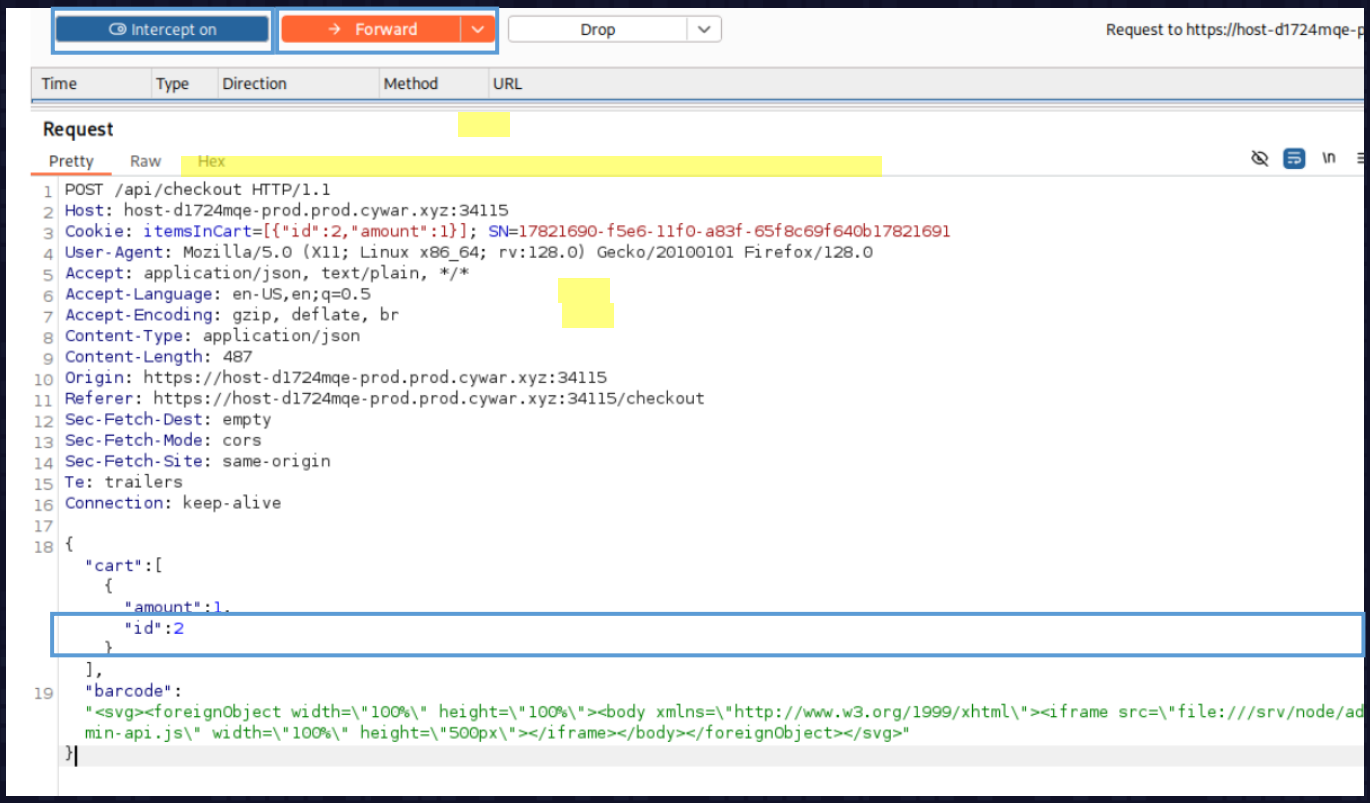


Figure 5: Intercepted checkout request using Burpsuite and modified the barcode by crafting an SVG structure that will retrieve a file from the server, based on the information of the existence of the file called “admin-api.js”.

```
// [Yesterday 6:58 PM]
const config = {
        admin_token: 'dcb129fb258a43a525efaebb3dc7512d'
}

const getDB = (token) => {
        return new MySQL.connection(token).getDB()
}

const checkAuth = (token) => {
        const db = getDB(token)

        if (db.auth(token)) {
                return console.warn("Success")
        } else {
                return console.warn("Unauthorized")
        }
}

function request(config) {
        if (!checkAuth(admin_token)) {
                logout()
        } else {
                new Request(config)
        }
}

const main = () => {
        request({ ...config, method: "GET", url:
"/admin/users" })
}

console.warn(config.admin_token)
```

Figure 6: Upon checking the result on the website we receive a pdf which contains Javascript code from “admin-api.js” and in that code we can see an admin token.

Remediation Options

- Enforce proper input validation and sanitization for SVG and markup content - https://tinyurl.com/frd6x98a
- Apply network-level egress filtering to restrict outbound requests from application servers - https://tinyurl.com/fr2mwwfz
- Isolate sensitive internal files and services from web-facing components - https://tinyurl.com/3vs3z2y7

VULN-002 Improper Validation of Specified Quantity in Input (Medium)

Description

The product receives input that is expected to specify a quantity (such as size or length), but it does not validate or incorrectly validates that the quantity has the required properties.

Specified quantities include size, length, frequency, price, rate, number of operations, time, and others. Code may rely on specified quantities to allocate resources, perform calculations, control iteration, etc.

Reference - https://tinyurl.com/253zysfx

Details

During the assessment, an improper input validation weakness was identified in the handling of item quantities within the shopping and checkout process. The issue was observed in the amount parameter, which was included in requests related to the cart contents and the purchase confirmation workflow.

By manipulating the amount value during the checkout process, the application accepted invalid numeric input without enforcing logical boundaries. Negative values were processed by the server and reflected directly in the generated receipt, resulting in a negative total purchase price.

This behaviour demonstrates a lack of server-side validation for user-supplied quantities, allowing the purchase logic to be abused. The vulnerability was accessible to any user capable of completing a purchase and did not require elevated privileges. The impact was confirmed through the receipt output, indicating potential for repeated financial manipulation if abused.

Notes

Portswigger's Burpsuite tool was used - https://tinyurl.com/4k4hbfv5

Burp Suite is a powerful tool for intercepting and analysing HTTP requests between the client and server. By configuring the browser to use Burp's proxy, all web traffic is captured in the Proxy > HTTP history tab. Captured requests can then be sent to the Repeater tab for detailed inspection.

Figure 1: Initial cart interaction prior to exploitation

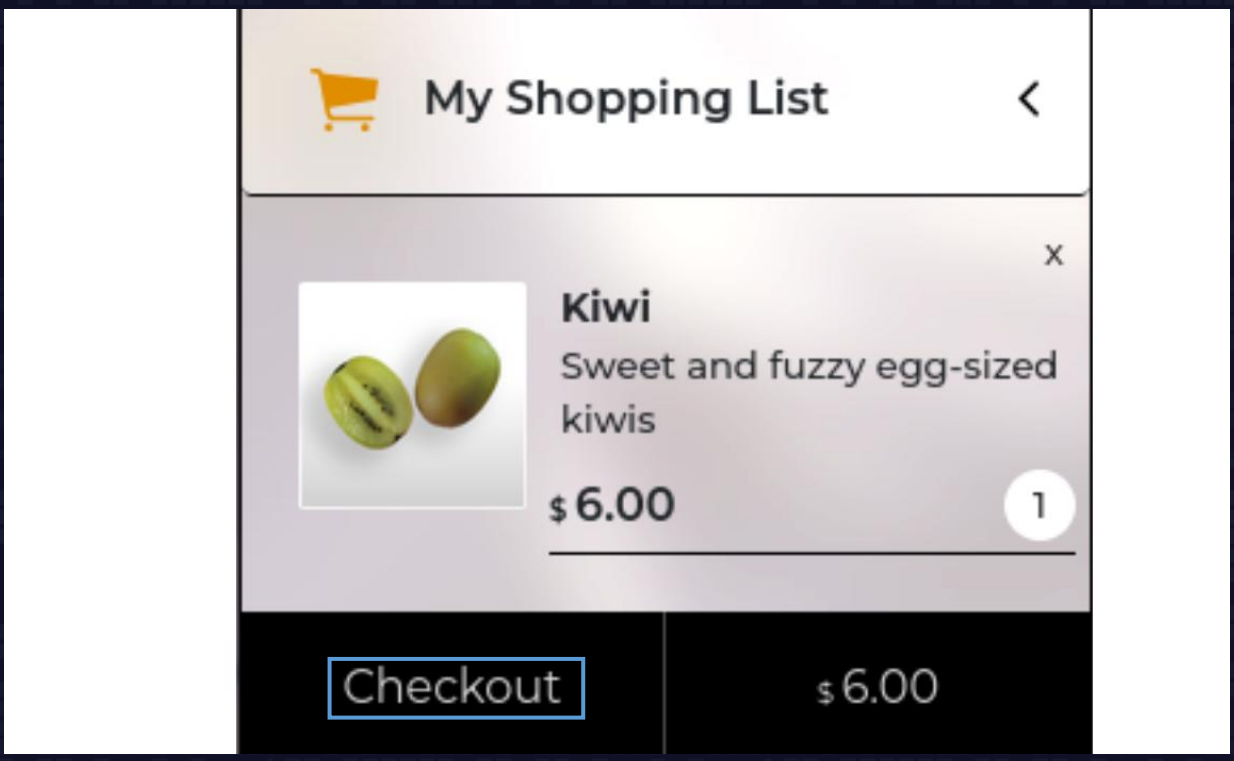


Figure 2: Cart inspection and transition to checkout

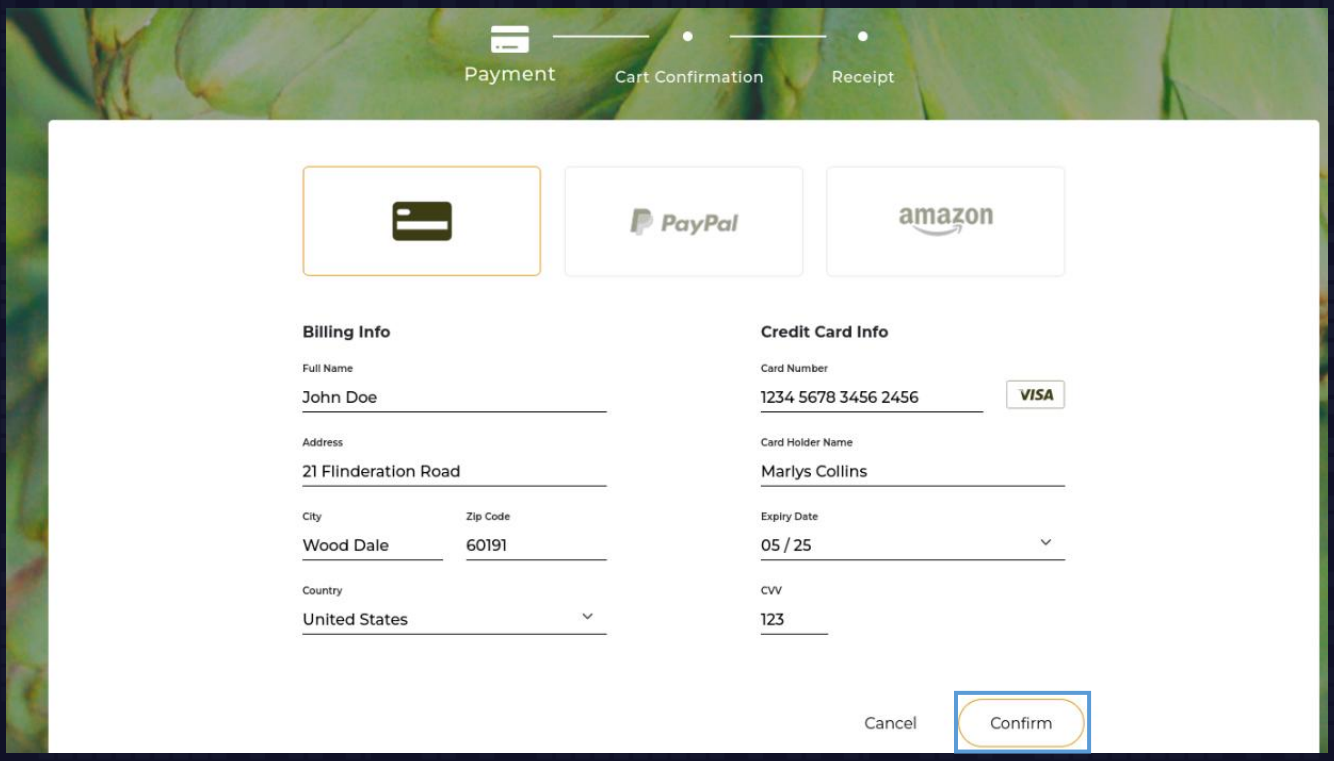


Figure 3: Billing confirmation

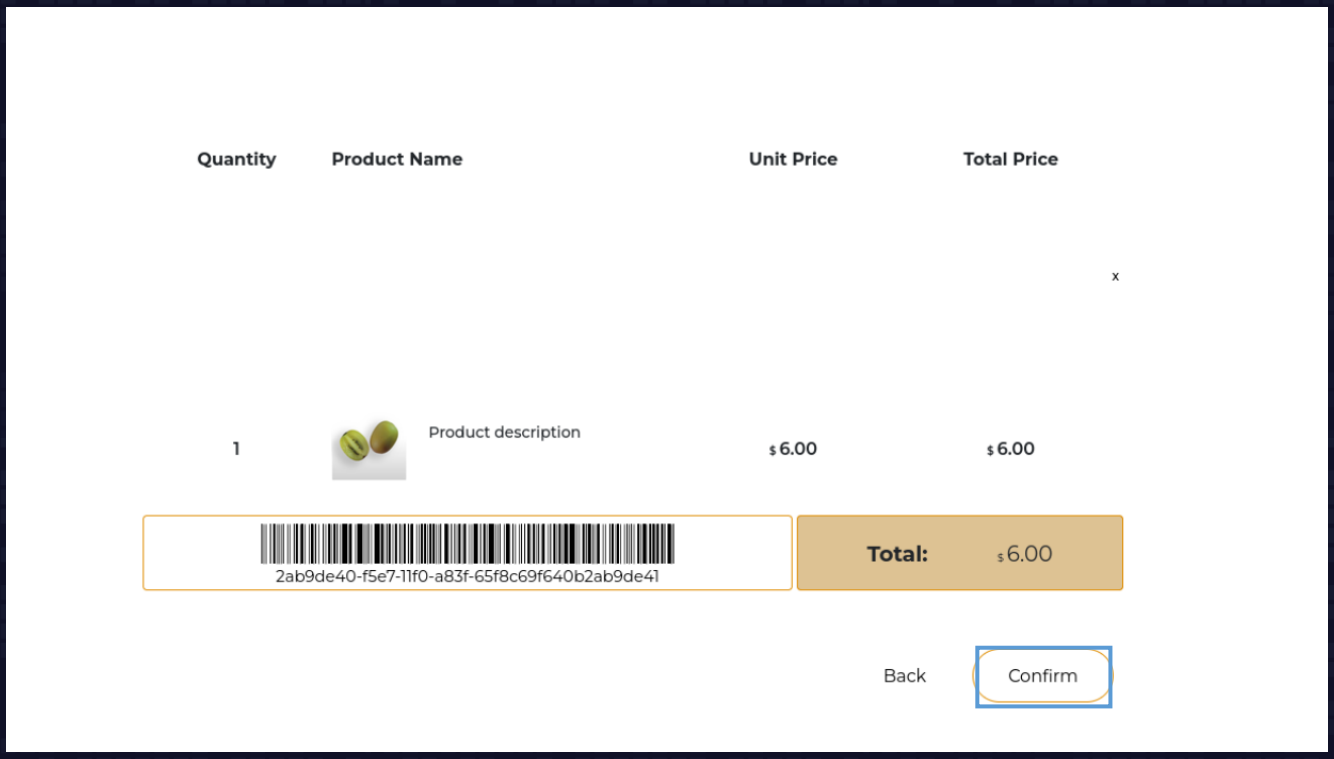


Figure 4: Purchase confirmation which will send the request to the server

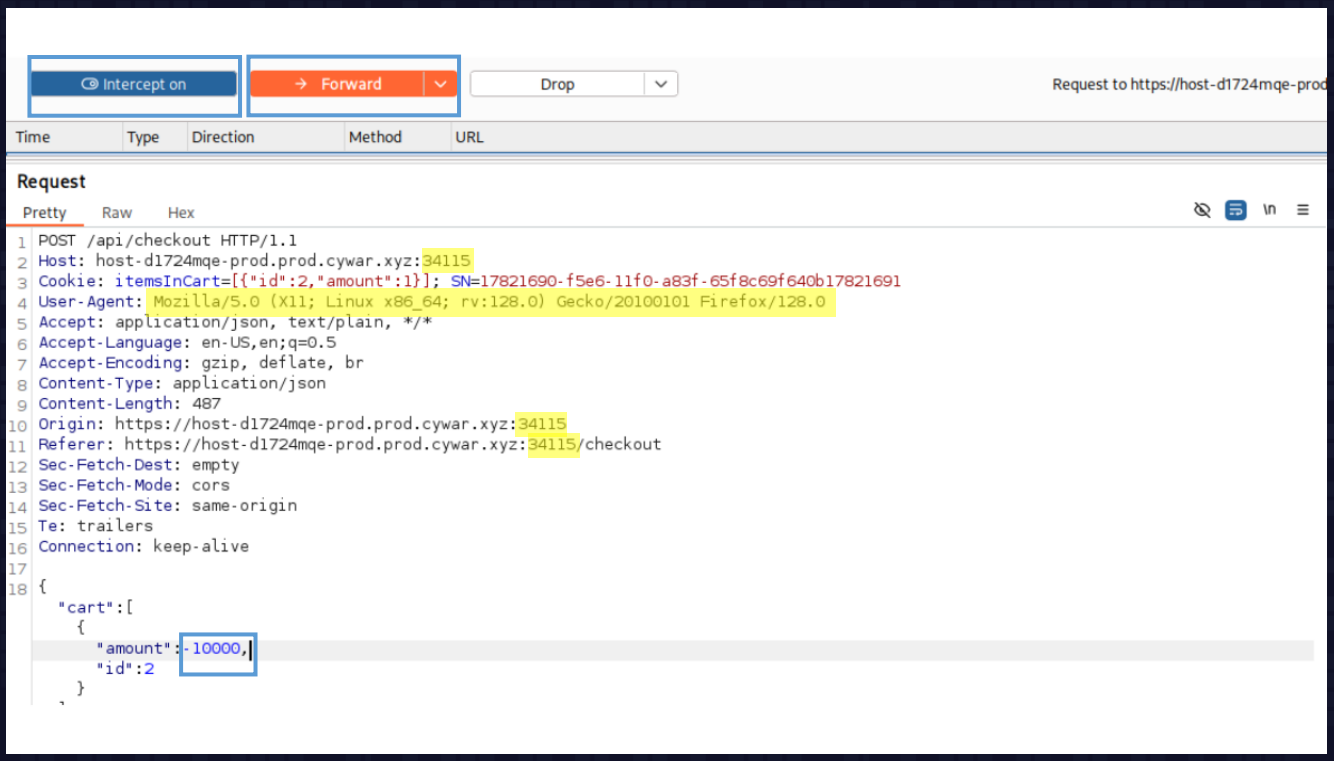


Figure 5: Intercepting the request using Burpsuite, I change the amount to a negative value and forward the request to the server

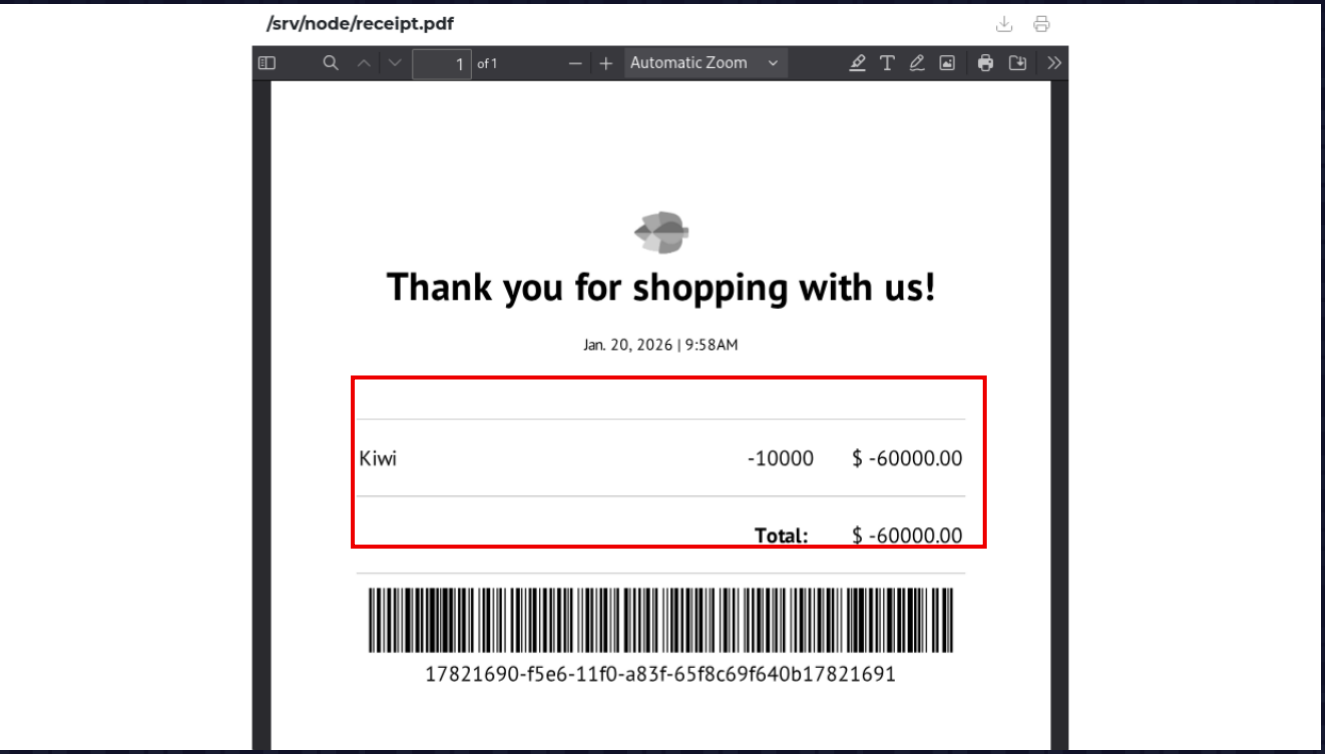


Figure 6: The response from the server is a receipt and as shown the amount was not validated resulting in a negative price for the order

Remediation

- Enforce strict server-side validation for numeric input - https://tinyurl.com/ybynkzpm
- Perform server-side recalculation of totals instead of trusting client-supplied values - https://tinyurl.com/bdwswec7