Critical SSRF: Internal Administrative Bypass via Stock API Manipulation

Project Information

- **Analyst:** Mduduzi William Radebe
- **Date:** 17 January 2026
- **Platform:** PortSwigger Web Security Academy
- **Vulnerability Type:** CWE-918: Server-Side Request Forgery (SSRF)
- **Severity: Critical (9.1/10)**

1. Executive Summary

A critical **Server-Side Request Forgery (SSRF)** vulnerability was identified in the application's stock-check functionality. The server was found to trust a user-supplied URL parameter to fetch data from back-end systems without proper validation. By manipulating this parameter, I forced the server to make requests to its own internal administrative interface (localhost). This allowed me to bypass authentication entirely and perform unauthorized administrative actions, including user deletion.

2. Technical Vulnerability Breakdown

The Attack Surface

The application features a "Check Stock" button that communicates with a back-end API. The request uses a stockApi parameter to specify the source of the data:

POST /product/stock

stockApi=http://stock.weliketoshop.net:8080/details?productId=1

The Underlying Flaw

The application exhibits **Insecure Trust in User Input**. It takes the URL provided in the stockApi parameter and executes a server-side request. Because the internal admin panel (/admin) is configured to allow all traffic from 127.0.0.1 (localhost) without a password, the web server acts as an unintentional "proxy" for the attacker.

3. Exploitation Methodology (Proof of Concept)

Step 1: Traffic Analysis

I intercepted the stock check request using **Burp Suite**. I observed that the stockApi value was a full URL, suggesting the server was fetching this content server-side.

Step 2: Testing for Localhost Access

Using Burp Repeater, I replaced the legitimate API URL with the internal loopback address:

stockApi=http://localhost/admin

The server responded with an **HTTP 200 OK** and the HTML source of the administration page. This confirmed that the server-side request was successful and that the admin panel lacked secondary authentication for local requests.

Step 3: Internal Reconnaissance

I reviewed the rendered HTML from the localhost/admin request and identified the specific endpoint for user management:

<a href="/admin/delete?username=carlos">Delete</a>

Step 4: Executing the Unauthorized Action

I crafted a final SSRF payload to trigger the deletion of the carlos account. By submitting this through the stockApi parameter, the **server itself** made the request to the delete endpoint.

Final Payload:

stockApi=http://localhost/admin/delete?username=carlos

Step 5: Verification

The server's response confirmed the deletion was successful. I had successfully performed a **Critical administrative action** without ever being prompted for a username or password.

4. Root Cause Analysis

Component	**Failure**	**Impact**
Input Validation	The stockApi parameter accepts any URL string.	Allows targeting of internal IP addresses and local services.
Trust Model	The internal /admin panel trusts 127.0.0.1 implicitly.	Bypasses the need for administrative credentials.
Network Architecture	Lack of segmentation between public web logic and internal admin logic.	Facilitates lateral movement within the infrastructure.

5. Remediation & Hardening Recommendations

A. Implement an Allowlist (Primary Defense)

The application should never allow a user to specify an arbitrary URL. Instead, the stockApi parameter should be validated against a strict **allowlist** of permitted domains (e.g., stock.weliketoshop.net only).

B. Block Access to Loopback & Private IP Ranges

The back-end code should be configured to reject any request where the destination is localhost, 127.0.0.1, or private IP ranges (RFC 1918) such as 10.0.0.0/8 or 192.168.0.0/16.

C. Enforce Authentication on All Interfaces

Even if a request originates from localhost, the administrative panel should still require a strong session token or password. "Security through Obscurity" (hiding the panel on an internal IP) is not a substitute for robust authentication.

6. Conclusion

The identification of this SSRF vulnerability proves that the application's internal security relies on the false assumption that internal services are unreachable. By turning the web server against itself, I achieved a total compromise of the administrative interface. Implementing an allowlist-based approach for all server-side requests is the most effective way to mitigate this risk.