Penetration Test Report

[subtitle]

0xe10c@hack.me

2020-07-25

Contents

1 Executive Summary

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Tortor at auctor urna nunc id cursus. Sed vulputate mi sit amet mauris commodo. In ante metus dictum at tempor commodo ullamcorper a lacus. Suspendisse faucibus interdum posuere lorem ipsum dolor.

2 Report

2.1 Enumeration

2.1.1 TCP services - common

```
nmap -sC -sV -oA nmap/jupiter 10.10.11.216 # scan for top 1000 default ports
```

```
┌──(kali㉿kali)-[~/htb/jupiter]
└─$ nmap -sC -sV -oA nmap/jupiter 10.10.11.216
Starting Nmap 7.94 ( https://nmap.org ) at 2023-06-16 12:53 EDT
Nmap scan report for 10.10.11.216
Host is up (0.030s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 ac:5b:be:79:2d:c9:7a:00:ed:9a:e6:2b:2d:0e:9b:32 (ECDSA)
|_  256 60:01:d7:db:92:7b:13:f0:ba:20:c6:c9:00:a7:1b:41 (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://jupiter.htb/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.06 seconds
```

redirect leaks domain

Figure 2.1: nmap scan leaking domain name

An nmap scan of the 1000 most common TCP ports reveals a web server running on Port 80, with a leaked domain name of `jupiter.htb`. Accessing the website, it's a web sever hosting information about space.

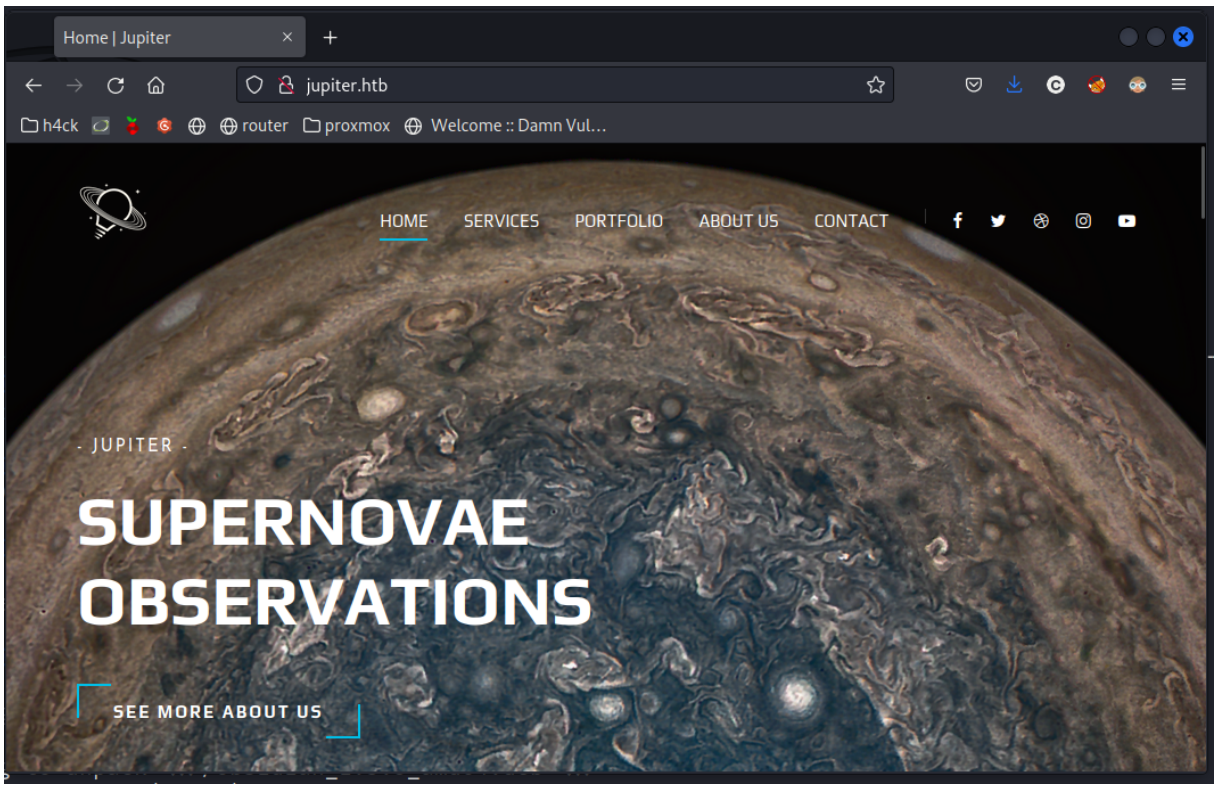

Figure 2.2: jupiter home page

2.2 Subdomain: kiosk.jupiter.htb

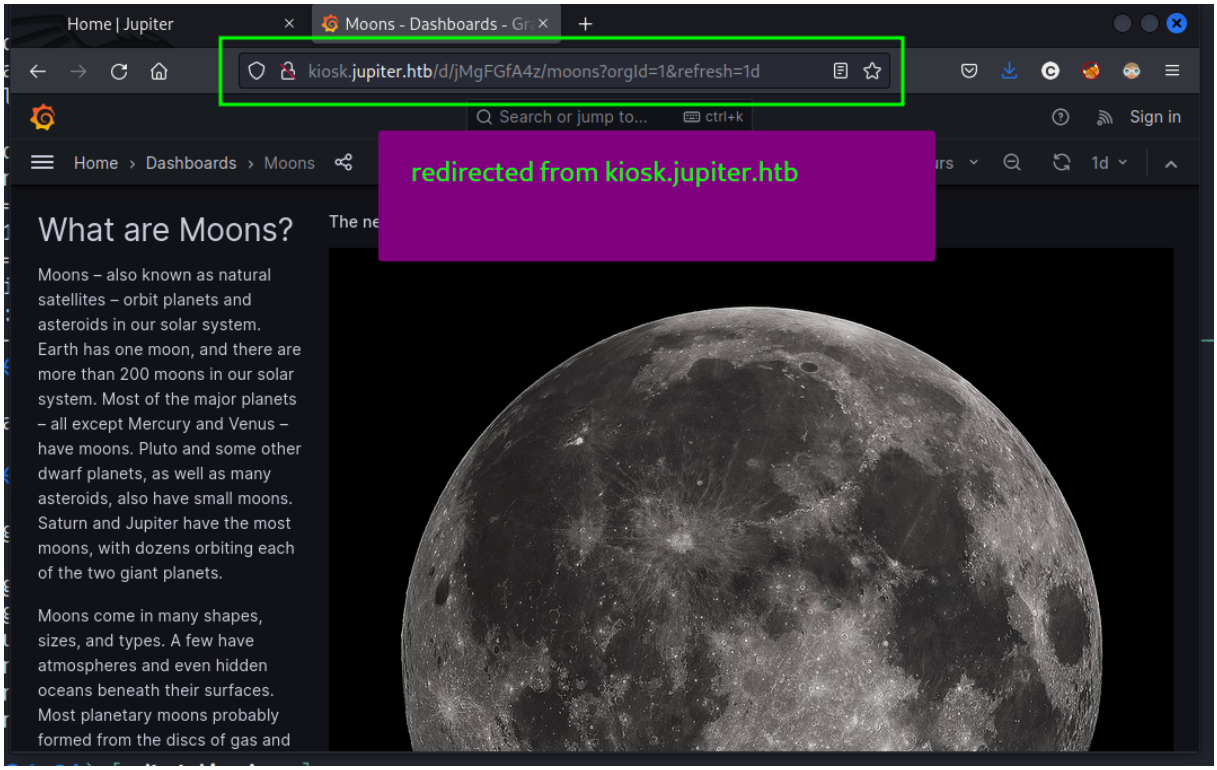

Figure 2.3: landing page for kiosk subdomain

Accessing `kiosk.jupiter.htb`, there is a rapid succession of redirects, landing on the page shown above. Scrolling down the page, information populates dynamically on the page suggesting the use of a back-end database.Examining the requests, eventually we find a POST request to the `/api/ds/query` API endpoint, transmitting JSON data with a field named `rawSql`, which is vulnerable to SQL injection.

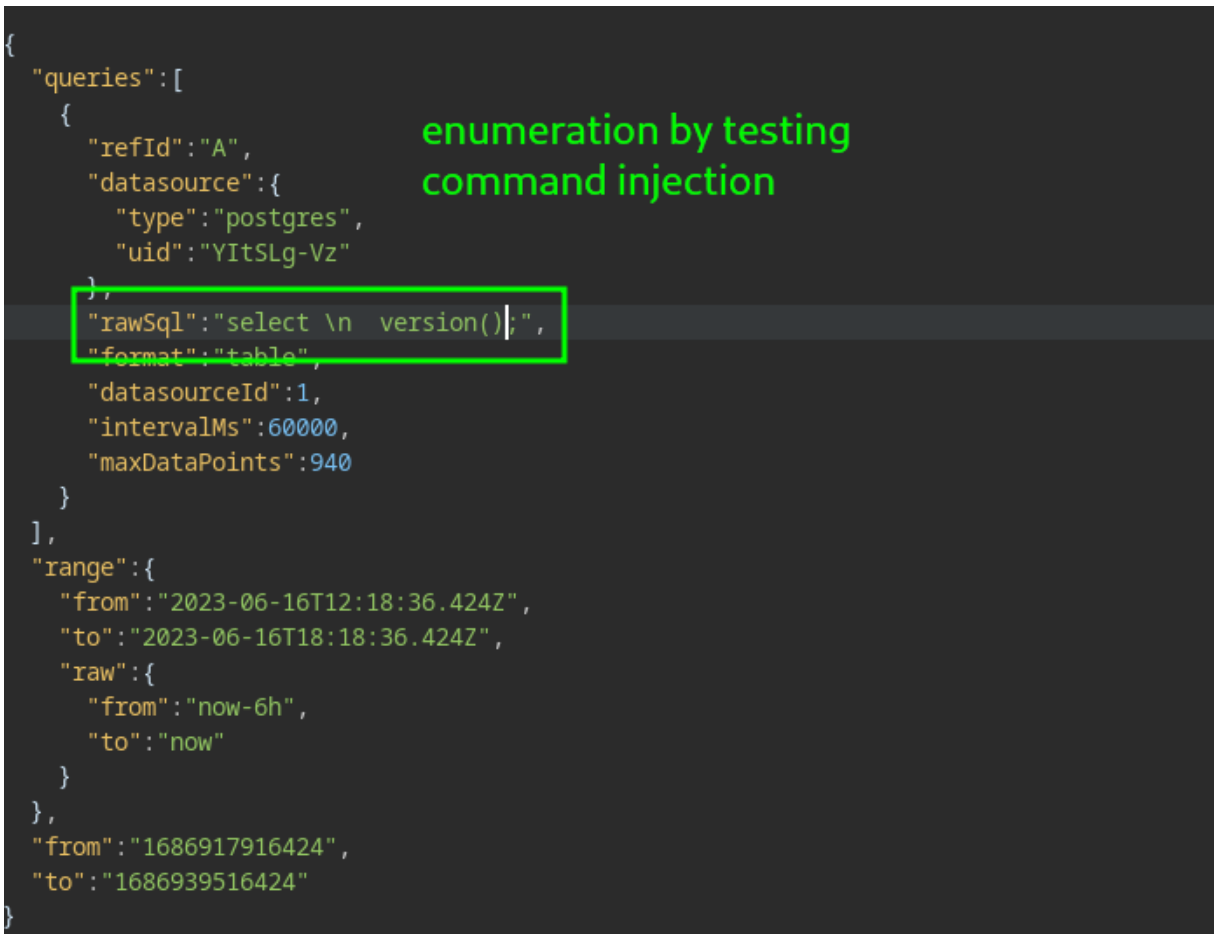

Figure 2.4: vulnerable sql parameter

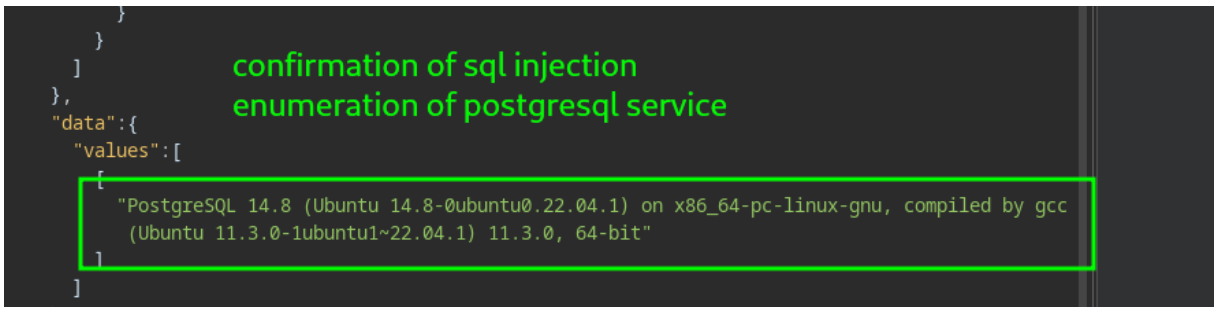

Figure 2.5: confirming sql injection vulnerability

2.3 Initial Foothold: `sqlmap --os-shell`

Using the saved request, we can gain a rudimentary web shell with the command `sqlmap --os-shell` as the `postgres` user. This is a low-privilege user who has read/write access on the system, as well as networking.

Using this web shell, transitioned to a more traditional reverse shell, enabling persistence through the use of the `cron` scheduling utility to send a reverse shell back to the attacking machine once every minute.#### more ports The command `ss -ltp` reveals a few listening ports which did not show up in the nmap scan.

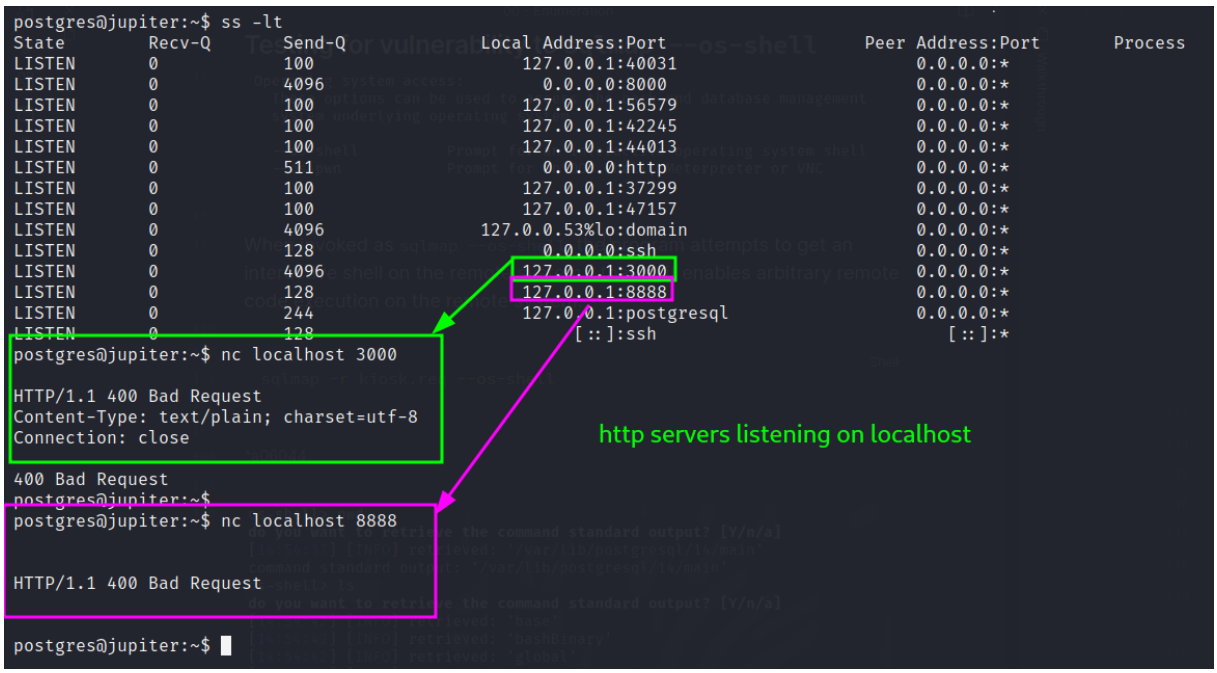

Figure 2.6: additional services on the target

2.3.1 users with shell access

```
postgres@jupiter:~$ grep sh$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
juno:x:1000:1000:juno:/home/juno:/bin/bash
postgres:x:114:120:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
jovian:x:1001:1002:,,,:/home/jovian:/bin/bash
postgres@jupiter:~$
```

four users with shell access

Figure 2.7: users with shell access

We can see in the `/etc/passwd` file there are four users on the target with shell access: * `root` * `postgres` * `juno` * `jovian`

2.4 Pivot to `juno` user

`juno` belongs to the groups `juno` and `science`.

2.4.1 files owned by `juno` user

Using the command below, we discovered a number of files owned by the `juno` user in the `/dev/shm` directory.

```
find / -user juno -type f -ls 2>/dev/null
```

```
postgres@jupiter:~$ id juno
uid=1000(juno) gid=1000(juno) groups=1000(juno),1001(science)
postgres@jupiter:~$
postgres@jupiter:~$ find / -user juno -type f -ls 2>/dev/null
     3385      4 -rw-rw-r--   1 juno     juno         1929 Jun 22 21:08 /dev/shm/shadow.data/sim-stats.json
     3359      4 -rw-rw-r--   1 juno     juno         1889 Jun 22 21:08 /dev/shm/shadow.data/processed-config.yaml
     3384      0 -rw-rw-r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/server/server.python3.10.1000.exitcode
     3367      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/server/server.python3.10.1000.shimlog
     3366      4 -rw-r--r--   1 juno     juno          177 Jun 22 21:08 /dev/shm/shadow.data/hosts/server/server.python3.10.1000.stderr
     3365      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/server/server.python3.10.1000.stdout
     3382      4 -rw-rw-r--   1 juno     juno            1 Jun 22 21:08 /dev/shm/shadow.data/hosts/client3/client3.curl.1000.exitcode
     3375      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/client3/client3.curl.1000.shimlog
     3374      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/client3/client3.curl.1000.stderr
     3373      4 -rw-r--r--   1 juno     juno          548 Jun 22 21:08 /dev/shm/shadow.data/hosts/client3/client3.curl.1000.stdout
     3381      4 -rw-rw-r--   1 juno     juno            1 Jun 22 21:08 /dev/shm/shadow.data/hosts/client2/client2.curl.1000.exitcode
     3371      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/client2/client2.curl.1000.shimlog
     3370      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/client2/client2.curl.1000.stderr
     3369      4 -rw-r--r--   1 juno     juno          548 Jun 22 21:08 /dev/shm/shadow.data/hosts/client2/client2.curl.1000.stdout
     3383      4 -rw-rw-r--   1 juno     juno            1 Jun 22 21:08 /dev/shm/shadow.data/hosts/client1/client1.curl.1000.exitcode
     3379      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/client1/client1.curl.1000.shimlog
     3378      0 -rw-r--r--   1 juno     juno            0 Jun 22 21:08 /dev/shm/shadow.data/hosts/client1/client1.curl.1000.stderr
     3377      4 -rw-r--r--   1 juno     juno          548 Jun 22 21:08 /dev/shm/shadow.data/hosts/client1/client1.curl.1000.stdout
     2225      4 -rw-rw-rw-   1 juno     juno          815 Mar  7 12:28 /dev/shm/network-simulation.yml
     4756    160 -rw-r-----   1 juno     juno       163509 Jun 22 18:04 /var/crash/_usr_bin_bash.1000.crash
```

juno has lots of stuff in /dev/shm

Figure 2.8: replace me 21

These files pertain to the Shadow Network Simulator.

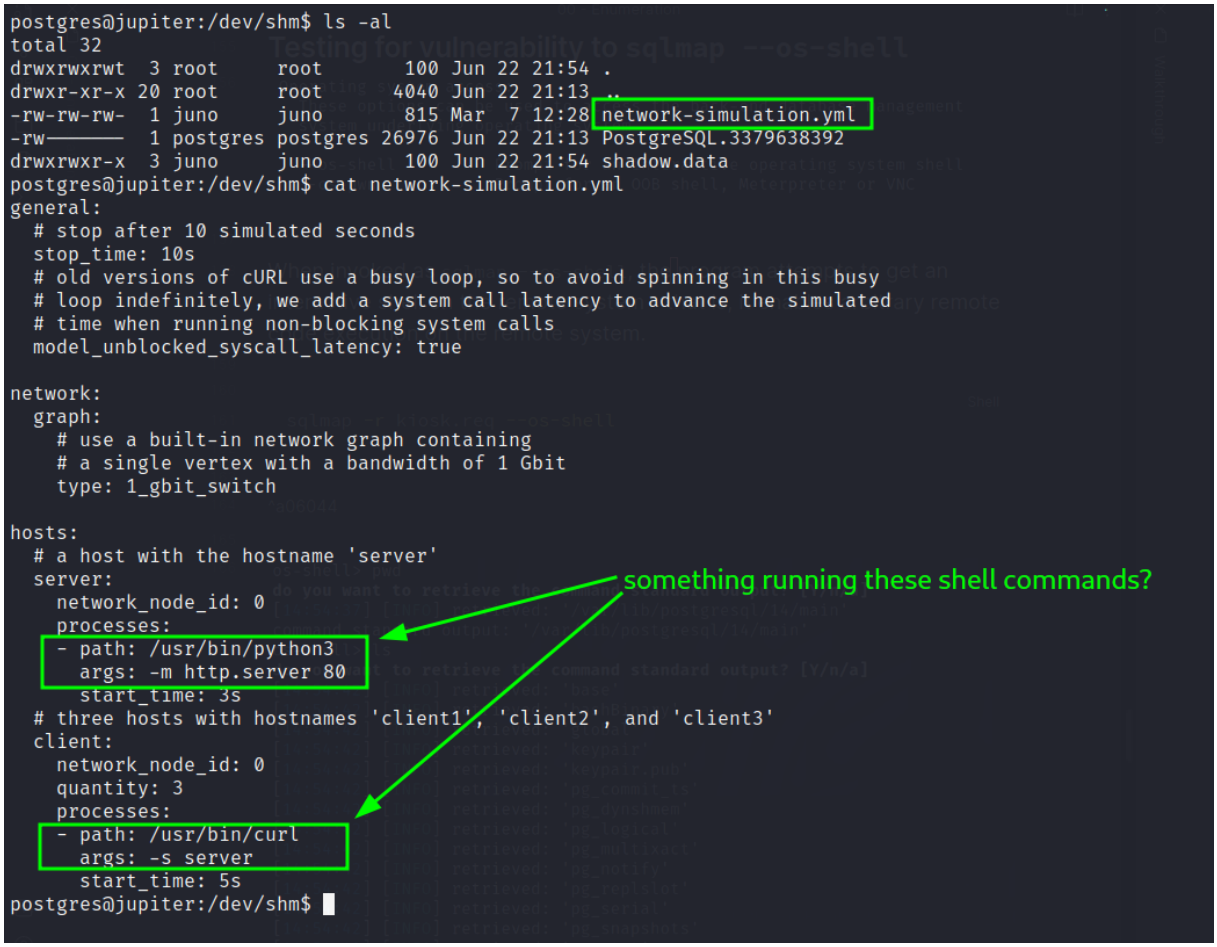

Figure 2.9: shadow network simulator configuration file

Searching the internet for “shadow network simulation”, we eventually come to this page describing the “non security” of the Shadow Network simulation tool:

> Shadow currently doesn’t restrict access to the host file system. A malicious managed program can read and modify the same files that Shadow itself can.

This configuration file can be abused to gain code execution as the `juno` user through the use of a copy of `bash` with the SUID bit set.

With this shell, we can manipulate `juno`’s SSH settings to gain a full shell as the `juno` user, and find the user flag.#### user flag

```
juno@jupiter:~$ id
uid=1000(juno) gid=1000(juno) groups=1000(juno),1001(science)
juno@jupiter:~$ ls -al
total 52
drwxr-x---  8 juno juno 4096 May  4 12:10 .
drwxr-xr-x  4 root root 4096 Mar  7 13:00 ..
lrwxrwxrwx  1 juno juno    9 Mar  7 10:45 .bash_history -> /dev/null
-rw-r--r--  1 juno juno  220 Jan  6  2022 .bash_logout
-rw-r--r--  1 juno juno 3792 Mar  7 10:00 .bashrc
drwx------  3 juno juno 4096 May  4 18:59 .cache
drwxrwxr-x  5 juno juno 4096 Mar  7 10:02 .cargo
drwxrwxr-x  5 juno juno 4096 Mar  7 12:08 .local
-rw-r--r--  1 juno juno  828 Mar  7 10:00 .profile
drwxrwxr-x  6 juno juno 4096 Mar  7 10:01 .rustup
drwxrwxr-x 12 juno juno 4096 Mar  9 10:31 shadow
-rwxrwxr-x  1 juno juno  174 Apr 14 14:28 shadow-simulation.sh
drwx------  2 juno juno 4096 Jun 23 01:54 .ssh
-rw-r-----  1 root juno   33 Jun 22 23:43 user.txt
juno@jupiter:~$ cat user.txt
[redacted]a3cf9
juno@jupiter:~$
```

Figure 2.10: user flag

2.5 SSH port forwarding - investigating other listening services

```
postgres@jupiter:~$ ss -lt
State       Recv-Q      Send-Q           Local Address:Port           Peer Address:Port      Process
LISTEN      0           100                  127.0.0.1:40031               0.0.0.0:*
LISTEN      0           4096                   0.0.0.0:8000                0.0.0.0:*
LISTEN      0           100                  127.0.0.1:56579               0.0.0.0:*
LISTEN      0           100                  127.0.0.1:42245               0.0.0.0:*
LISTEN      0           100                  127.0.0.1:44013               0.0.0.0:*
LISTEN      0           511                    0.0.0.0:http                0.0.0.0:*
LISTEN      0           100                  127.0.0.1:37299               0.0.0.0:*
LISTEN      0           100                  127.0.0.1:47157               0.0.0.0:*
LISTEN      0           4096              127.0.0.53%lo:domain             0.0.0.0:*
LISTEN      0           128                    0.0.0.0:ssh                 0.0.0.0:*
LISTEN      0           4096                 127.0.0.1:3000                0.0.0.0:*
LISTEN      0           128                  127.0.0.1:8888                0.0.0.0:*
LISTEN      0           244                  127.0.0.1:postgresql          0.0.0.0:*
LISTEN      0           128                     [::]:ssh                      [::]:*
postgres@jupiter:~$ nc localhost 3000

HTTP/1.1 400 Bad Request
Content-Type: text/plain; charset=utf-8
Connection: close

400 Bad Request
postgres@jupiter:~$
postgres@jupiter:~$ nc localhost 8888


HTTP/1.1 400 Bad Request

postgres@jupiter:~$
```

http servers listening on localhost

Figure 2.11: additional services on the target

We can use SSH port forwarding to access the other ports on the machine.

2.5.1 Port 3000

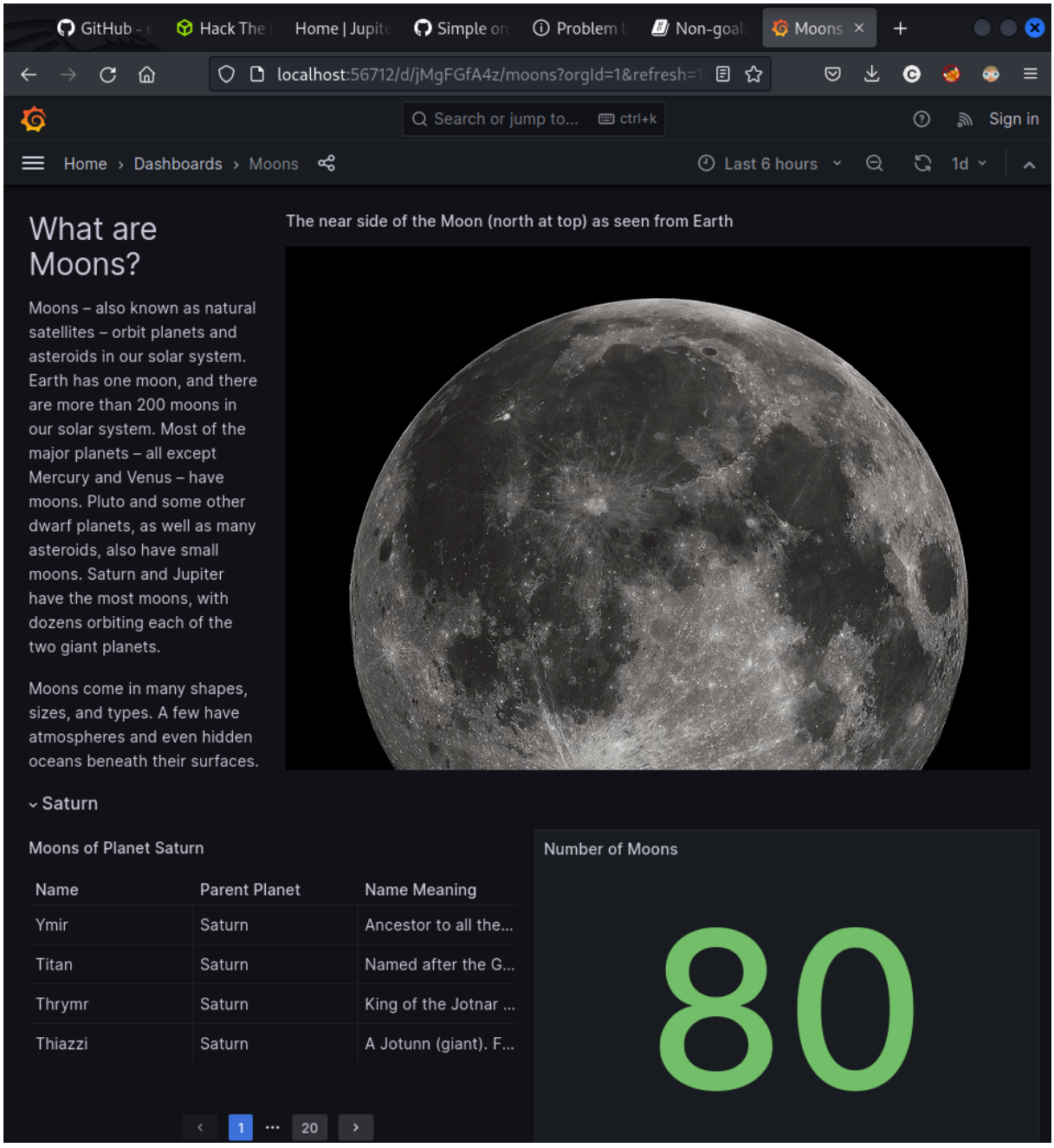

Figure 2.12: original jupiter dot htb website

Port 3000 is serving the `jupiter.htb` website.

2.5.2 Port 8888

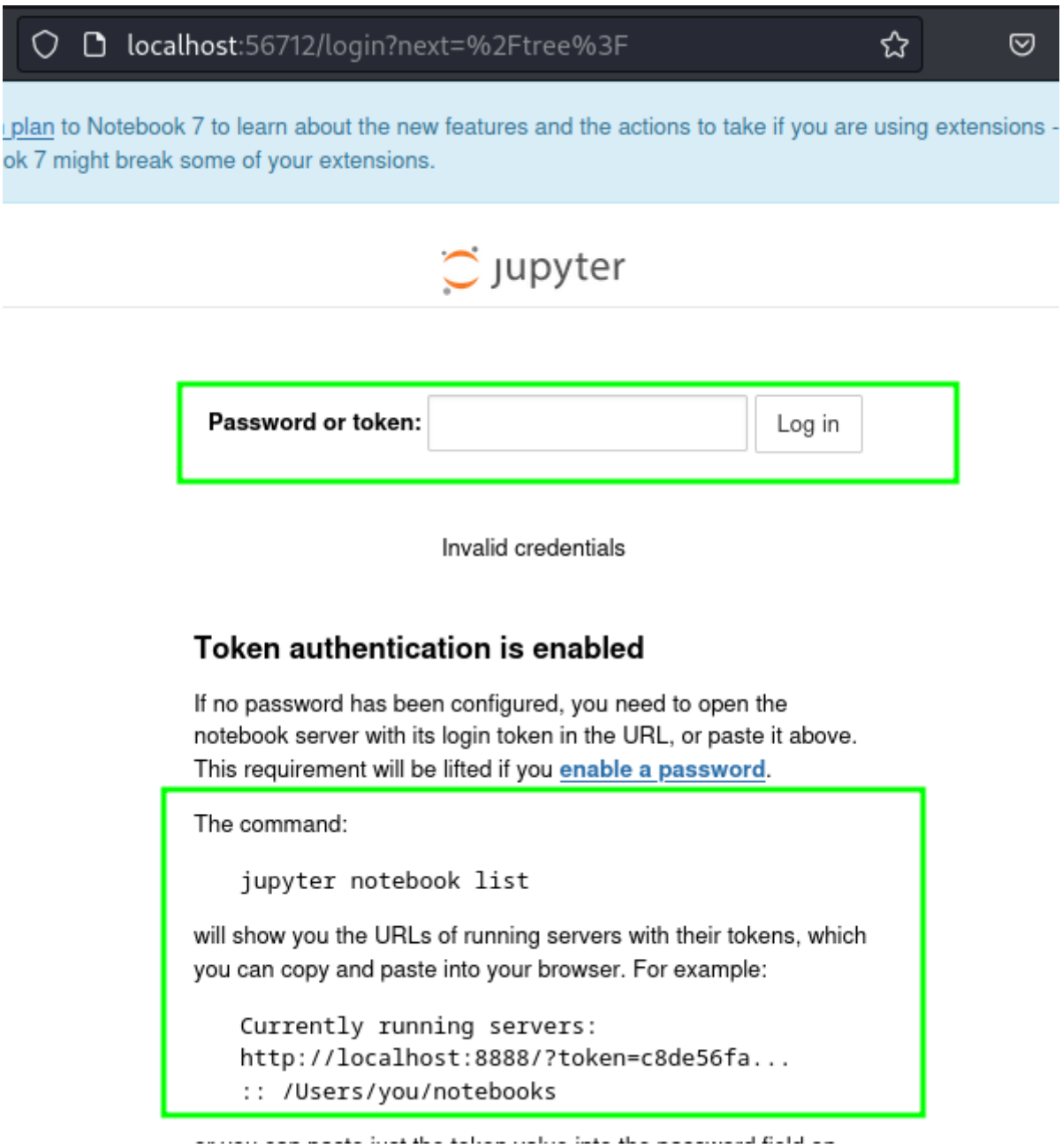

Figure 2.13: jupyter notebook instance

As `juno`, attempts to use the `jupyter notebook list` command fail:

```
juno@jupiter:~$ jupyter notebook list
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-notebook", line 5, in <module>
    from notebook.notebookapp import main
  File "/usr/local/lib/python3.10/dist-packages/notebook/notebookapp.py", line 80, in <module>
    from .services.contents.manager import ContentsManager
  File "/usr/local/lib/python3.10/dist-packages/notebook/services/contents/manager.py", line 17, in <module>
    from nbformat import sign, validate as validate_nb, ValidationError
  File "/usr/local/lib/python3.10/dist-packages/nbformat/__init__.py", line 11, in <module>
    from . import v1, v2, v3, v4
  File "/usr/local/lib/python3.10/dist-packages/nbformat/v4/__init__.py", line 40, in <module>
    from .convert import downgrade, upgrade
  File "/usr/local/lib/python3.10/dist-packages/nbformat/v4/convert.py", line 13, in <module>
    from .. import validator
  File "/usr/local/lib/python3.10/dist-packages/nbformat/validator.py", line 16, in <module>
    from .json_compat import ValidationError, _validator_for_name, get_current_validator
  File "/usr/local/lib/python3.10/dist-packages/nbformat/json_compat.py", line 11, in <module>
    import jsonschema
ModuleNotFoundError: No module named 'jsonschema'
juno@jupiter:~$
```

command fails as juno

Figure 2.14: attempting jupyter notebook command

If we look to see which processes are running under `jovian`, we can see a Python call associated with the Jupyter Notebook instance, running out of `/opt/solar-flares/`.

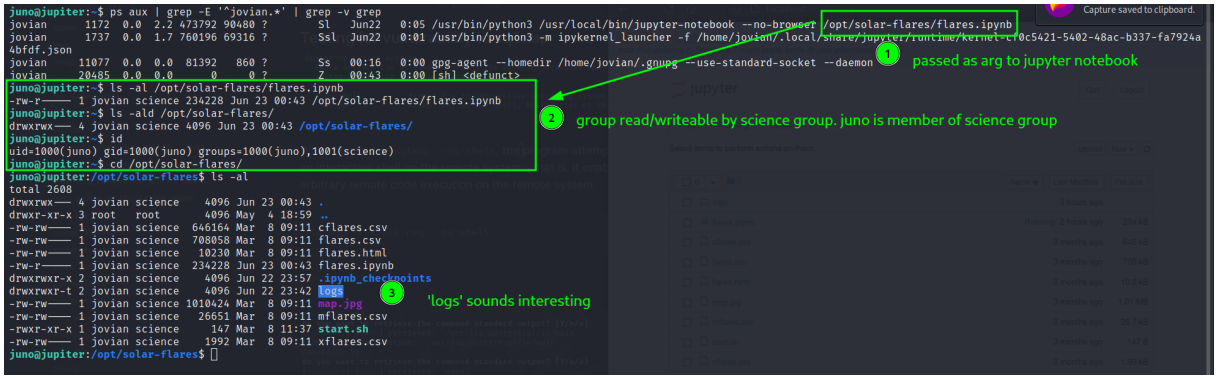

Figure 2.15: jupyter notebook process information

The argument passed to Jupyter is group read/writable by members of the `science` group, which includes `juno`. The `logs` directory in this folder contains many log files dating back months. Examining them exposes tokens used to access the Jupyter notebook instance.

```
[W 23:42:51.859 NotebookApp] Terminals not available (error was No module named 'terminado')
[I 23:42:51.866 NotebookApp] Serving notebooks from local directory: /opt/solar-flares
[I 23:42:51.866 NotebookApp] Jupyter Notebook 6.5.3 is running at:
[I 23:42:51.866 NotebookApp] http://localhost:8888/?token=4c9229a18dfa86d907c3673917d23947504b137a1c28d6d2
[I 23:42:51.866 NotebookApp]  or http://127.0.0.1:8888/?token=4c9229a18dfa86d907c3673917d23947504b137a1c28d6d2
[I 23:42:51.866 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 23:42:51.870 NotebookApp] No web browser found: could not locate runnable browser.
[C 23:42:51.870 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/jovian/.local/share/jupyter/runtime/nbserver-1172-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=4c9229a18dfa86d907c3673917d23947504b137a1c28d6d2
     or http://127.0.0.1:8888/?token=4c9229a18dfa86d907c3673917d23947504b137a1c28d6d2
```

Figure 2.16: exposed jupyter notebook token

Supplying a token to the Jupyter notebook allows us to gain code execution as the `jovian` user and send a reverse shell back to the attacker.

2.6 reverse shell as `jovian`

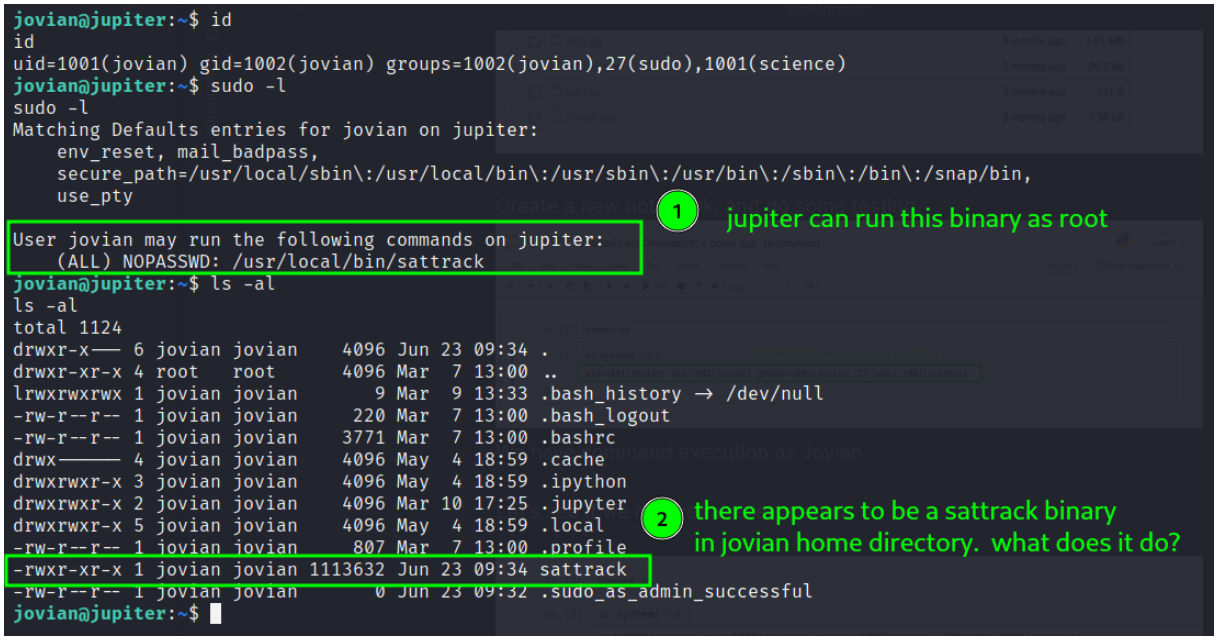

Figure 2.17: commands jovian can run as root

User `jovian` can run the binary `/usr/local/bin/sattrack` with root privileges, without the need to supply a password.

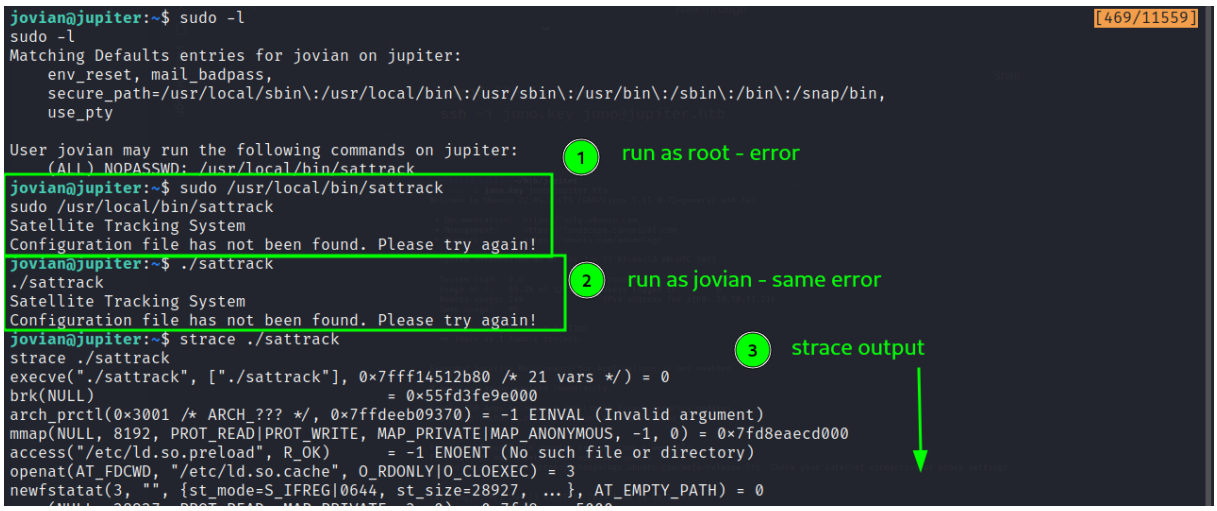

Figure 2.18: running sattrack binary

We get the same error when we run as root and as `jovian` - complaining about some config file not being found. Using `strace` we see that the program can't open the file `/tmp/config.json`.

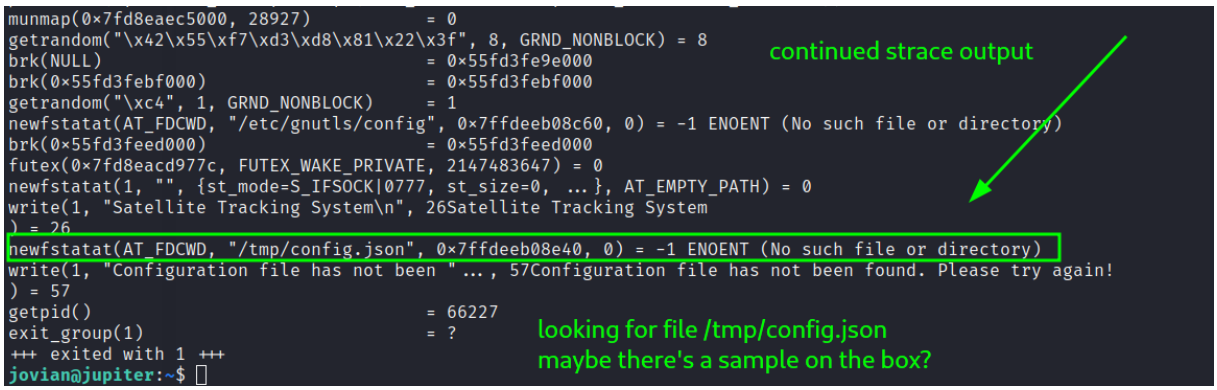

Figure 2.19: strace output leaking config.json file

2.7 Root Flag

When supplied with the file `/tmp/config.json`, the `sattrack` binary creates the `/tmp/tle` directory and writes to it. We can abuse the URI in the `tlesources` section to get local file inclusion (LIF) as `root`, and read the root flag `/root/root.txt`.

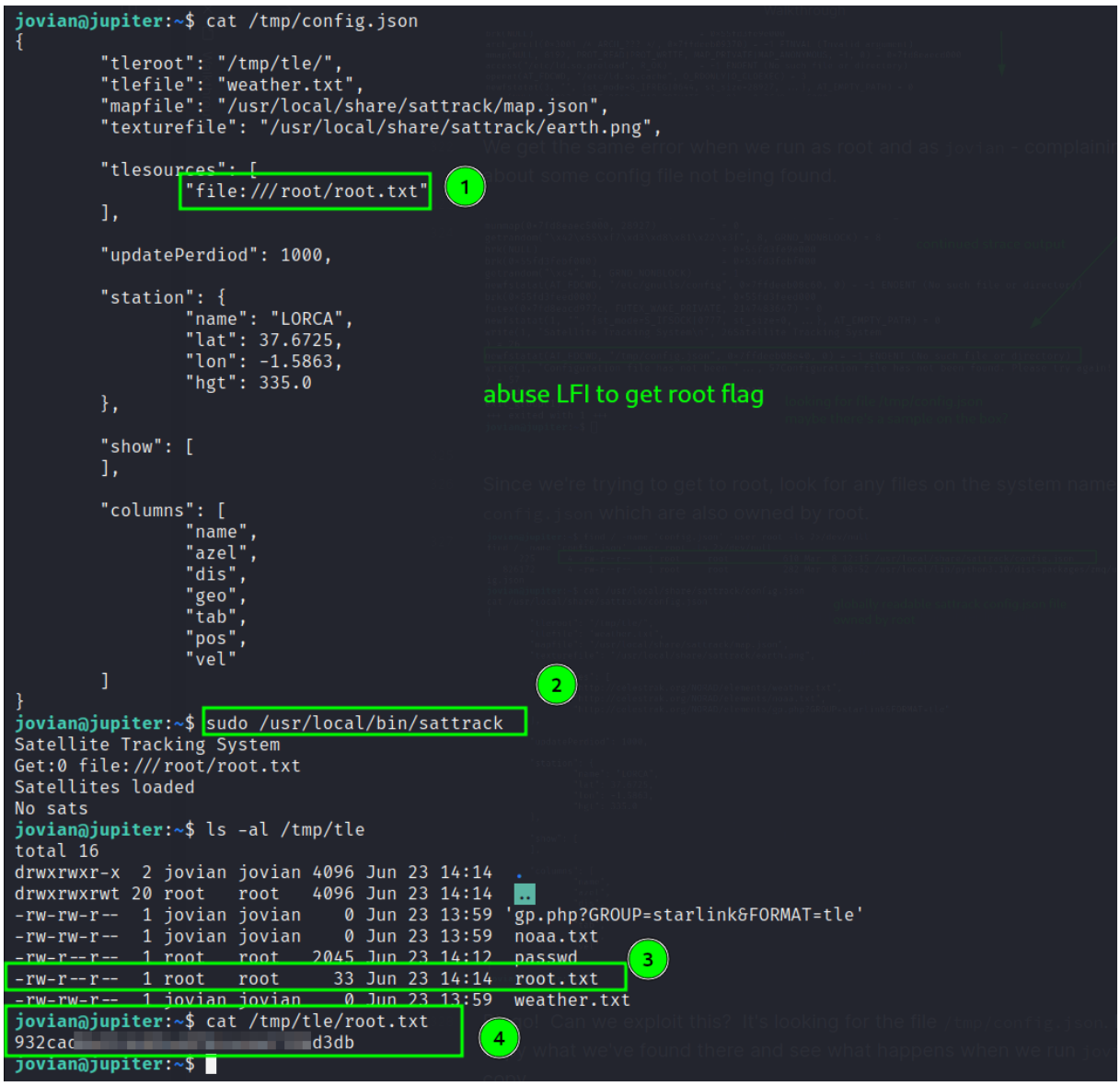

Figure 2.20: abusing LFI to read root flag