EMBEDI
(https://embedi.com)

Home (https://embedi.com) / Blog (https://embedi.com/blog) /
Research (https://embedi.com/blog/categories/research/) /
Enlarge your botnet with: top D-Link routers (DIR8xx D-Link routers cruisin' for a bruisin')

Categories:

- Analytics (https://embedi.com/blog/categories/analytics/)
- Research (https://embedi.com/blog/categories/research/)

Tags:

#ATM (https://embedi.com/blog/tags/atm/) #CISCO (https://embedi.com/blog/tags/cisco/)

#cybersecurity (https://embedi.com/blog/tags/cybersecurity/)

#D-Link (https://embedi.com/blog/tags/d-link/) #DJI (https://embedi.com/blog/tags/dji/)

#exploitation (https://embedi.com/blog/tags/exploitation/)

#firmware-security (https://embedi.com/blog/tags/firmware-security/)

#hardware (https://embedi.com/blog/tags/hardware/) #hijacking (https://embedi.com/blog/tags/hijacking/)

#intel (https://embedi.com/blog/tags/intel/) #Microsoft (https://embedi.com/blog/tags/microsoft/)

#mobile (https://embedi.com/blog/tags/mobile/) #Office (https://embedi.com/blog/tags/office/)

#RCE (https://embedi.com/blog/tags/rce/) #router (https://embedi.com/blog/tags/router/)

#SCADA (https://embedi.com/blog/tags/scada/)

#vulnerabilities (https://embedi.com/blog/tags/vulnerabilities/)

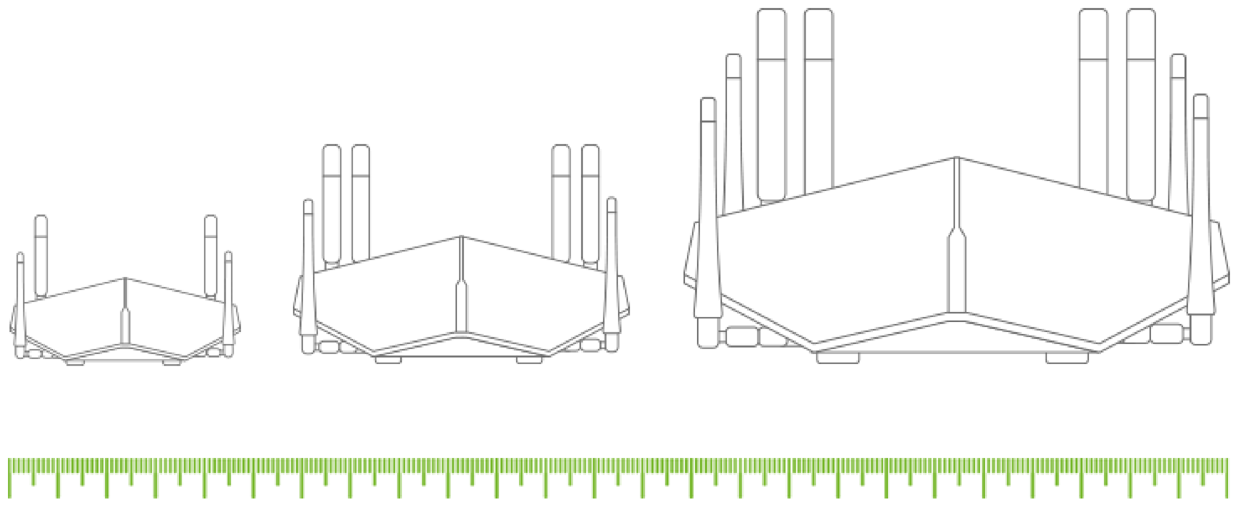
Popular articles:

Killchain of IoT Devices. Part 2 (https://embedi.com/blog/killchain-iot-devices-part-2/)

Killchain of IoT Devices. Part 1 (https://embedi.com/blog/killchain-iot-devices-part-1/)

12 September, 2017

Enlarge your botnet with: top D-Link routers (DIR8xx D-Link routers cruisin' for a bruisin')



Category: Research (https://embedi.com/blog/categories/research/)
Tags: #D-Link (https://embedi.com/blog/tags/d-link/), #router (https://embedi.com/blog/tags/router/), # vulnerabilities (https://embedi.com/blog/tags/vulnerabilities/)

In this article, we are going to discuss vulnerabilities detected in the top D-Link routers:

- DIR890L
- DIR885L
- DIR895L
- and other DIR8xx D-Link routers cruising for a bruising.

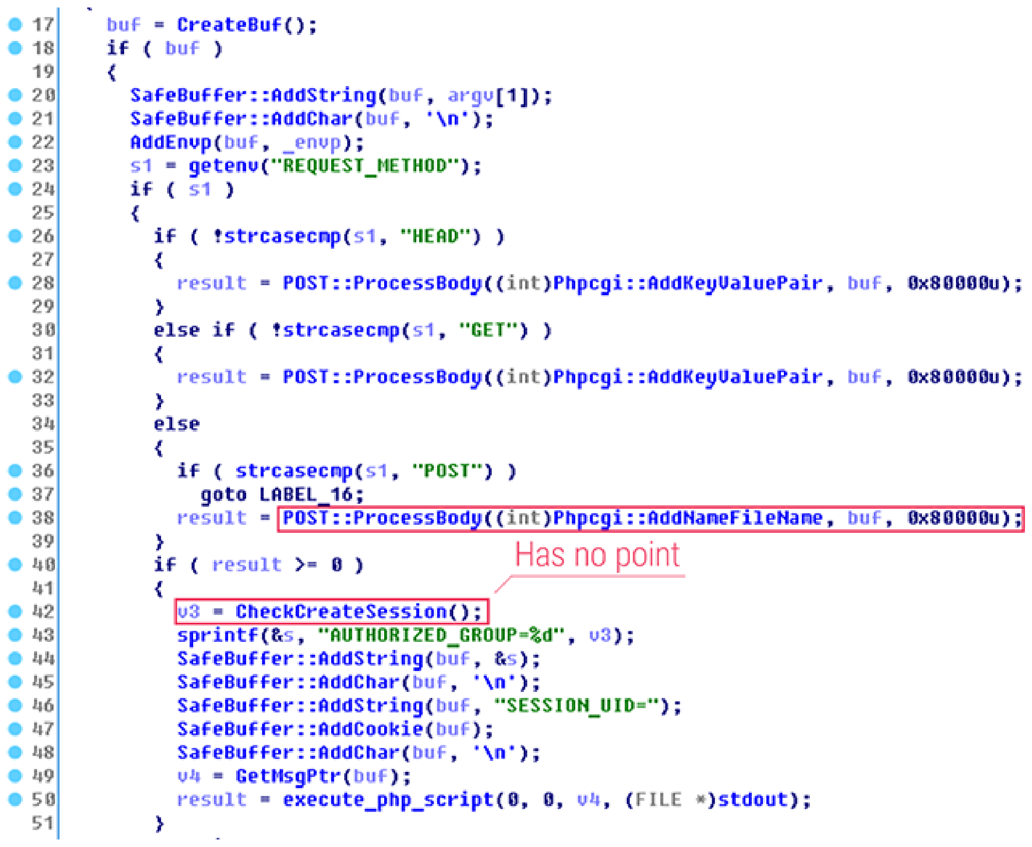
The devices use the same code, thus giving a magnificent and quite tempting opportunity to attackers to add them to a botnet. Moreover, we have managed to make Mirai for the devices by modifying its compilation script a bit.

We will also say a couple of words about our interaction with the developer (which has brought no results, while the vulnerabilities are still not closed). Two vulnerabilities are related to the cgibin – the main CGI file that generates web interface pages to control the router. The other vulnerability deals with system recovery.

Stealing login and password

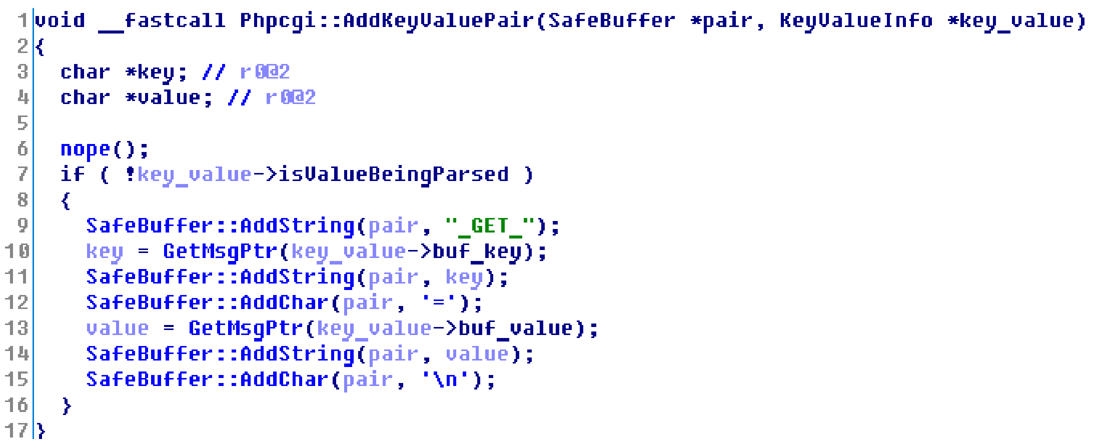
One HTTP request – login/password is in your bag.

The first detected vulnerability lies in phpcgi. Phpcgi is a symlink to cgibin and is responsible for processing requests to .php, .asp and .txt pages. It parses data sent via URL, HTTP headers or in the body of the POST request. phpcgi creates a long string which is later processed into sets of keys and values for the $_GET, $_POST, $_SERVER dictionaries as well as other variables of the php script. While finishing request analysis, the symlink checks user authorization. If a user is not authorized, it adds the AUTHORIZED_GROUP variable with -1 value to the string.



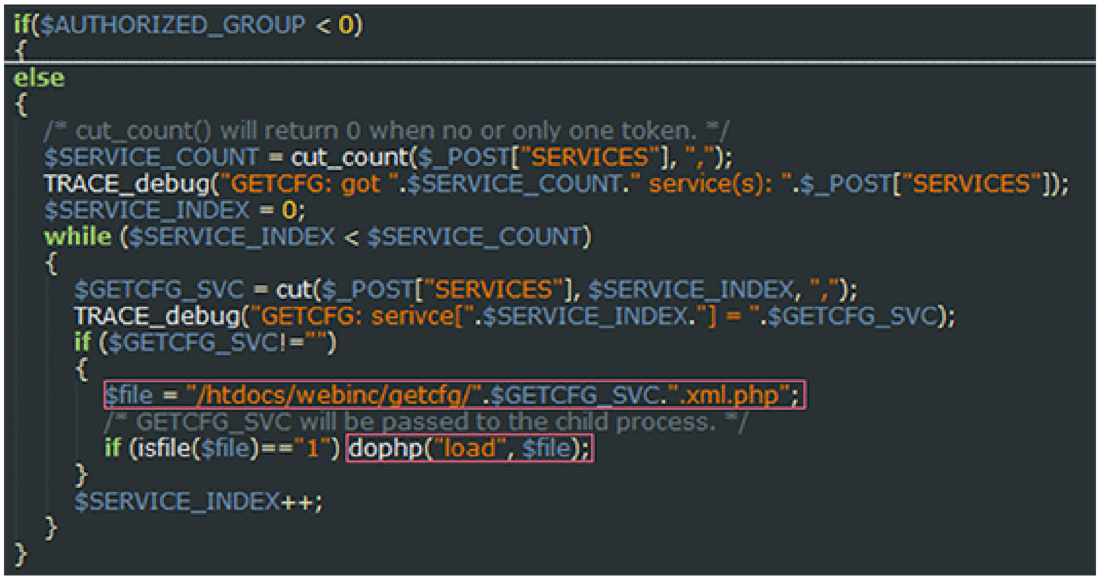
```
  buf = CreateBuf();
  if ( buf )
  {
    SafeBuffer::AddString(buf, argv[1]);
    SafeBuffer::AddChar(buf, '\n');
    AddEnvp(buf, _envp);
    s1 = getenv("REQUEST_METHOD");
    if ( s1 )
    {
      if ( !strcasecmp(s1, "HEAD") )
      {
        result = POST::ProcessBody((int)Phpcgi::AddKeyValuePair, buf, 0x80000u);
      }
      else if ( !strcasecmp(s1, "GET") )
      {
        result = POST::ProcessBody((int)Phpcgi::AddKeyValuePair, buf, 0x80000u);
      }
      else
      {
        if ( strcasecmp(s1, "POST") )
          goto LABEL_16;
        result = POST::ProcessBody((int)Phpcgi::AddNameFileName, buf, 0x80000u);
      }
      if ( result >= 0 )
      {
        v3 = CheckCreateSession();
        sprintf(&s, "AUTHORIZED_GROUP=%d", v3);
        SafeBuffer::AddString(buf, &s);
        SafeBuffer::AddChar(buf, '\n');
        SafeBuffer::AddString(buf, "SESSION_UID=");
        SafeBuffer::AddCookie(buf);
        SafeBuffer::AddChar(buf, '\n');
        v4 = GetMsgPtr(buf);
        result = execute_php_script(0, 0, v4, (FILE *)stdout);
      }
```

The problem here is that in terms of security the parsing process cannot be called flawless. Every key-value pair is encoded as follows: _TYPE_KEY=VALUE, where TYPE is GET, POST, SERVER or nothing. After that, the pairs are linked with the line break symbol '\n'.



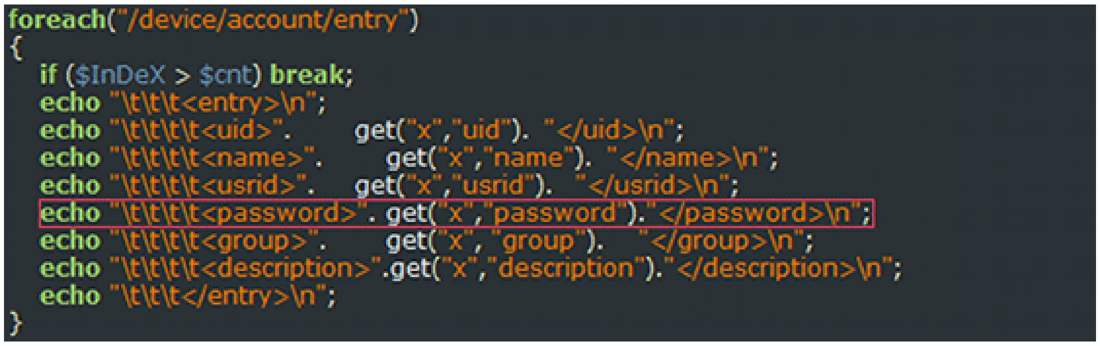
```
void __fastcall Phpcgi::AddKeyValuePair(SafeBuffer *pair, KeyValueInfo *key_value)
{
  char *key; // r0@2
  char *value; // r0@2

  nope();
  if ( !key_value->isValueBeingParsed )
  {
    SafeBuffer::AddString(pair, "_GET_");
    key = GetMsgPtr(key_value->buf_key);
    SafeBuffer::AddString(pair, key);
    SafeBuffer::AddChar(pair, '=');
    value = GetMsgPtr(key_value->buf_value);
    SafeBuffer::AddString(pair, value);
    SafeBuffer::AddChar(pair, '\n');
  }
}
```

By sending the POST request, we can add a key with the SomeValue{69c90e256be360ad980ca26f621f2003f22fc8a173440dabd89573a4d6bbc248}3dAUTHORIZED_GROUP=1 value. This key will be processed into _GET_SomeKey=SomeValue\nAUTHORIZED_GROUP=1, which will enable us to start scripts, though limited in their amount, as an authorized user. By sending a request to http://192.168.0.1/getcfg.php and adding the pair SERVICES=DEVICE.ACOUNT, we will call the /htdocs/webinc/getcfg/DEVICE.ACCOUNT.xml.php script that will return login and password from the router.



```
if($AUTHORIZED_GROUP < 0)
{
else
{
    /* cut_count() will return 0 when no or only one token. */
    $SERVICE_COUNT = cut_count($_POST["SERVICES"], ",");
    TRACE_debug("GETCFG: got ".$SERVICE_COUNT." service(s): ".$_POST["SERVICES"]);
    $SERVICE_INDEX = 0;
    while ($SERVICE_INDEX < $SERVICE_COUNT)
    {
        $GETCFG_SVC = cut($_POST["SERVICES"], $SERVICE_INDEX, ",");
        TRACE_debug("GETCFG: serivce[".$SERVICE_INDEX."] = ".$GETCFG_SVC);
        if ($GETCFG_SVC!="")
        {
            $file = "/htdocs/webinc/getcfg/".$GETCFG_SVC.".xml.php";
            /* GETCFG_SVC will be passed to the child process. */
            if (isfile($file)=="1") dophp("load", $file);
        }
        $SERVICE_INDEX++;
    }
}
```

It is quite evident that it enables cyber-criminals to start scripts stored in the /htdocs/webinc/getcfg folder. There is also the DEVICE.ACCOUNT.xml.php script in the given directory that can provide attackers with a good deal of critical information including a login and password to the device.



```
foreach("/device/account/entry")
{
    if ($InDeX > $cnt) break;
    echo "\t\t\t<entry>\n";
    echo "\t\t\t\t<uid>".      get("x","uid").  "</uid>\n";
    echo "\t\t\t\t<name>".      get("x","name").  "</name>\n";
    echo "\t\t\t\t<usrid>".    get("x","usrid").   "</usrid>\n";
    echo "\t\t\t\t<password>". get("x","password")."</password>\n";
    echo "\t\t\t\t<group>".      get("x", "group").     "</group>\n";
    echo "\t\t\t\t<description>".get("x","description")."</description>\n";
    echo "\t\t\t</entry>\n";
}
```

In other words, if attackers send a request to http://192.168.0.1/getcfg.php and add the SERVICES=DEVICE.ACOUNT pair, the device will respond with the page containing a login and password to the device.

That is more than enough for attackers to, for example, use their custom malicious firmware to update the device.

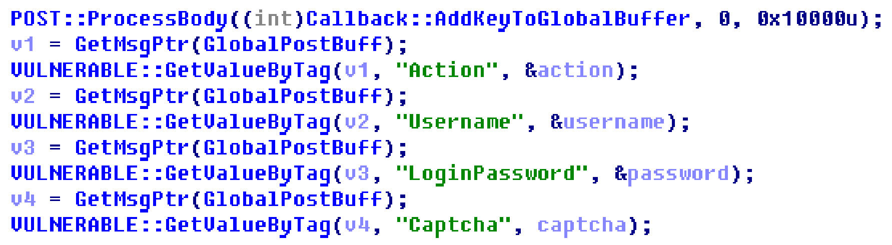
Exploit for the phpcgi vulnerability (https://github.com/embedi/DIR8xx_PoC/blob/master/phpcgi.py)

Full Superuser access (RCE to Root) to the device

Get root-shell with an HTTP request.

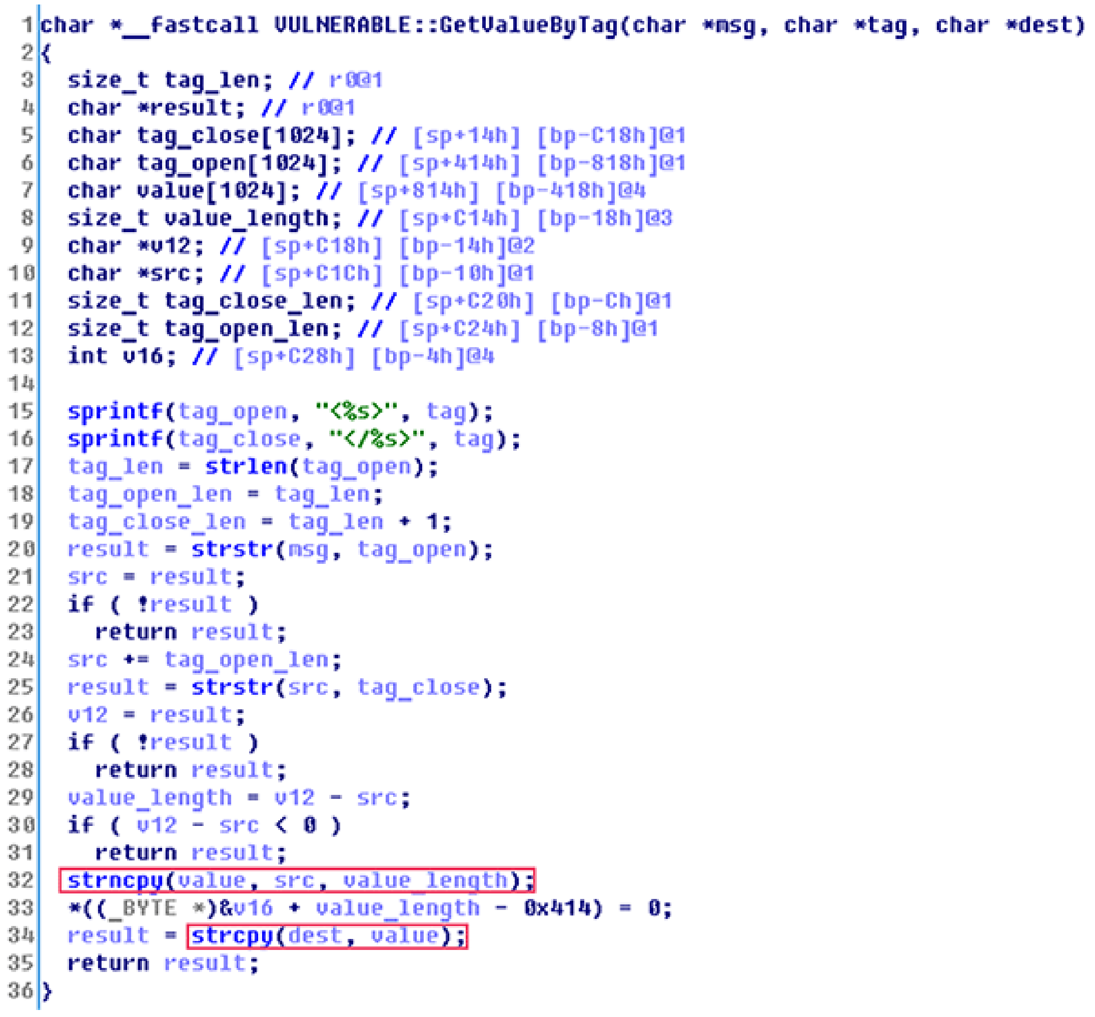
The second vulnerability is a stack overflow vulnerability caused by an execution error [HNAP (https://en.wikipedia.org/wiki/Home_Network_Administration_Protocol)].

To send a message via the protocol, an attacker sends a request to the http://192.168.0.1/HNAP1/ page and specifies the type of the request in the SOAPACTION header. It is the procedure of authorization request processing that is vulnerable. The "http://purenetworks.com/HNAP1/Login" value is used to call the authorization function. It is possible to specify different key-value pairs in a request body (there is a defined set of used keys): Action, Username, LoginPassword, and Captcha. After that these keys are encoded with html tags. E.g.: Here is the value



```
POST::ProcessBody((int)Callback::AddKeyToGlobalBuffer, 0, 0x10000u);
v1 = GetMsgPtr(GlobalPostBuff);
VULNERABLE::GetValueByTag(v1, "Action", &action);
v2 = GetMsgPtr(GlobalPostBuff);
VULNERABLE::GetValueByTag(v2, "Username", &username);
v3 = GetMsgPtr(GlobalPostBuff);
VULNERABLE::GetValueByTag(v3, "LoginPassword", &password);
v4 = GetMsgPtr(GlobalPostBuff);
VULNERABLE::GetValueByTag(v4, "Captcha", captcha);
```

The main problem here is the function that extracts key-value pairs, while there is a designated buffer in the stack with the size of 0x400 byte. Nonetheless, attackers can send up to 0x10000 bytes of data with the help of strncpy, which causes a huge stack overflow. Strncpy overflows the current stack and then "shoots a security shot" overflowing the calling function stack as well, because variable 'dest' may hold up to 0x80 bytes, while there are 0x400 bytes for a value.



```
char *__fastcall VULNERABLE::GetValueByTag(char *msg, char *tag, char *dest)
{
  size_t tag_len; // r0@1
  char *result; // r0@1
  char tag_close[1024]; // [sp+14h] [bp-C18h]@1
  char tag_open[1024]; // [sp+414h] [bp-818h]@1
  char value[1024]; // [sp+814h] [bp-418h]@4
  size_t value_length; // [sp+C14h] [bp-18h]@3
  char *v12; // [sp+C18h] [bp-14h]@2
  char *src; // [sp+C1Ch] [bp-10h]@1
  size_t tag_close_len; // [sp+C20h] [bp-Ch]@1
  size_t tag_open_len; // [sp+C24h] [bp-8h]@1
  int v16; // [sp+C28h] [bp-4h]@4

  sprintf(tag_open, "<%s>", tag);
  sprintf(tag_close, "</%s>", tag);
  tag_len = strlen(tag_open);
  tag_open_len = tag_len;
  tag_close_len = tag_len + 1;
  result = strstr(msg, tag_open);
  src = result;
  if ( !result )
    return result;
  src += tag_open_len;
  result = strstr(src, tag_close);
  v12 = result;
  if ( !result )
    return result;
  value_length = v12 - src;
  if ( v12 - src < 0 )
    return result;
  strncpy(value, src, value_length);
  *((_BYTE *)&v16 + value_length - 0x414) = 0;
  result = strcpy(dest, value);
  return result;
}
```

In addition, when exiting the function, in the R0 registry there is a pointer to the string. Thus, it is possible to specify an sh command set, change a return address to a 'system' function. Then, the only thing attackers have to do is to decide what they want to do with the controlled device.

Exploit for the HNAP vulnerability (https://github.com/embedi/DIR8xx_PoC/blob/master/hnap.py)

Updating firmware in Recovery mode.

One reboot and you're a root.

The third analyzed vulnerability is that, when the router is started, it sets up a recovery web server for several seconds. The server provides unauthorized cyber-criminals with an opportunity to update the device software, if they are connected to the device and, therefore, to a local network, with an Ethernet cable.

So, the only thing needed to exploit the vulnerability is to restart the device either by exploiting the vulnerabilities described above, or by sending the "EXEC REBOOT SYSTEM" command for service jcpd. This service is accessible from the local network via the 19541 port, can be easily used to restart the router without requiring authorization from a cyber-criminal, and is working permanently (no setting can turn it off). To get complete control over the device, an attacker needs to upload a modified firmware to the router.

Exploit for the System Recovery vulnerability

(https://github.com/embedi/DIR8xx_PoC/blob/master/update.sh)

Research timeline

However, that is not it, and the communication with the D-Link security team deserves special attention. Please, find the timeline below:

04/26/2017 – We notified the developer about the vulnerabilities detected in the hnap protocol.

04/28/2017 – D-Link employees replied that the vulnerabilities had been already patched in the beta version of their firmware that could be found at support.dlink.com.
N.B. There was no notion about the firmware at the D-Link web site.

04/28/2017 – 05/03/2017 – We analyzed the firmware version D-Link referred to in their reply and found out that one of the vulnerabilities the developer were notified about was still not fixed.

05/03/2017 – 05/09/2017 – We managed to find another vulnerability in the firmware, informed D-Link and asked them how was it going with fixing the previous one. They answered that the very process of detecting, fixing and assessing a vulnerability takes some time.

06/01/2017 – We notified CERT about the vulnerabilities. Here the reply its representative sent us:

> *"Greetings,*
>
> *Thank you for your vulnerability report submission. After review, we have decided not to handle the case.*
>
> *We recommend continuing to work with the vendor before proceeding with public disclosure."*

6/02/2017 – For almost a month there was no word from D-Link. So, we decided that it was a high time to remedy the situation somehow. So, we warned D-Link that we would disclose the vulnerability to the public if they went on doing nothing about the problem.

06/06/2017 – The company replied with a description of its vulnerability response process and sent a beta firmware, which fixed the phpcgi vulnerability. The other vulnerability, reported to D-Link, was still ignored by the developer, though (perhaps, D-Link security team still believed that it was fixed by their beta-firmware).

Once again, we wrote to D-Link about the unpatched vulnerability. No wonder, there was no response. Here is the last message we have received from the developer:

> *Good Day,*
>
> *Our R&D is working on a solution for the report. Until they establish why it happens, a proposed solution, and the scope of the issue (if it effects other models) we won't generally discuss the report.*
>
> *I should have some updates early this week.*
>
> *I have no authority on how you conduct your work. Once we have a fix we announce/disclose the details support.dlink.com with accreditation to the 3rd party.*
>
> *Typically the cycle of fixes is a couple of weeks for beta you can validate. Once validated we will offer it to the public as a beta, then it will move on to long term QA as an RC to be released. A full release cycle will usually take up to 90 days.*
>
> *If you choose to publish sooner please provide a URL that we can reference for the report. If you choose to request a CVE-id that is all we will need to accredit your report.*

In the middle of August, we visited support.dlink.com and found out the developer uploaded the very same beta-firmware. 2 bugs out of 3 are still to be patched.

So, the bottom line of our research is:

- D-Link has closed one of the detected vulnerabilities in the DIR890L router only, leaving other devices unsafe.
- Two other vulnerabilities were (and are still) ignored by the developer.

Well done, D-Link!

(https://www.linkedin.com/shareArticle?mini=true&url=https://embedi.com/blog/enlarge-your-botnet-top-d-link-routers-dir8xx-d-link-routers-cruisin-bruisin/&title=Enlarge your botnet with: top D-Link routers (DIR8xx D-Link routers cruisin’ for a bruisin’))

(http://twitter.com/intent/tweet?url=https://embedi.com/blog/enlarge-your-botnet-top-d-link-routers-dir8xx-d-link-routers-cruisin-bruisin/&text=Enlarge your botnet with: top D-Link routers (DIR8xx D-Link routers cruisin’ for a bruisin’))

() 102

9