

信息

<https://avd.aliyun.com/detail?id=AVD-2022-26209>

中危 Totolink多款路由器命令注入漏洞 (CVE-2022-26209)

CVE编号	利用情况	补丁情况	披露时间
CVE-2022-26209	暂无	官方补丁	2022-03-18

漏洞描述

Totolink A3100R、Totolink A830R和Totolink A810R都是无线双频路由器。
Totolink多款路由器产品的setUploadSetting函数的FileName参数存在命令注入漏洞。攻击者可利用该漏洞执行任意命令。

受影响系统:

TOTOLINK TOTOLink A800R 4.1.2cu.5137_B20200730
TOTOLINK TOTOLink A3100R 4.1.2cu.5050_B20200504
TOTOLINK TOTOLink A830R 5.9c.4729_B20191112
TOTOLINK TOTOLink A950RG 4.1.2cu.5161_B20200903
TOTOLINK TOTOLink A3000RU 5.9c.5185_B20201128
TOTOLINK Totolink A10R 4.1.2cu.5182_B201026

解决建议

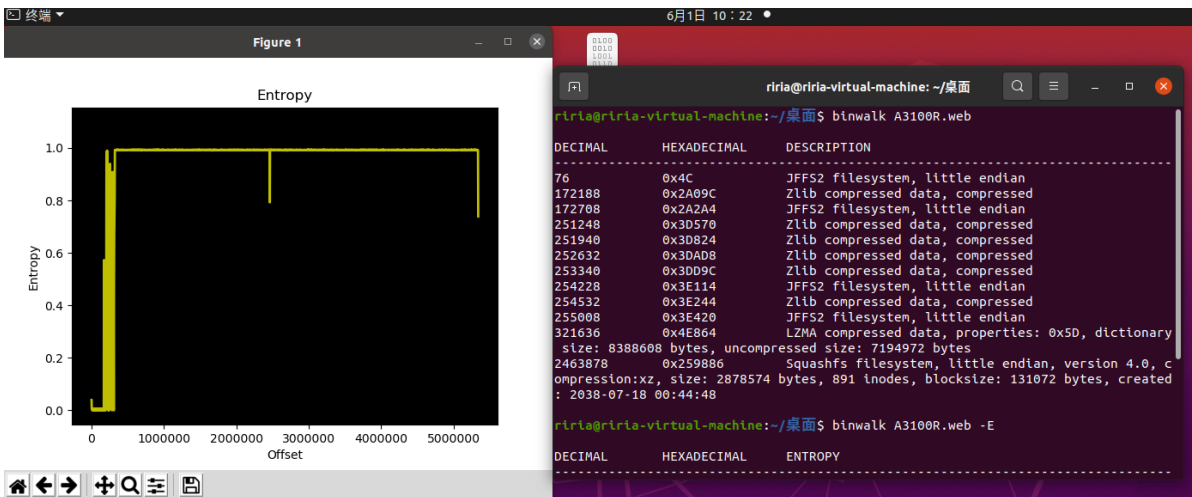
目前厂商已发布升级补丁以修复漏洞，补丁获取链接:

https://github.com/pjqwudi1/my_vuln/blob/main/totolink/vuln_24/24.md

参考链接

<https://cxsecurity.com/ckshow/CVE-2022-26209/>

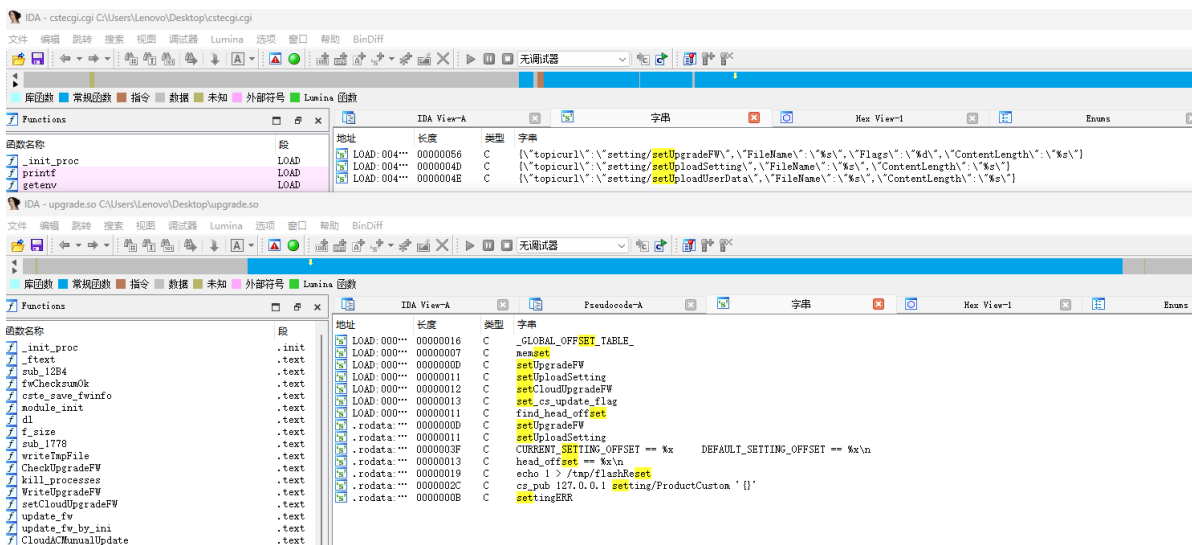
https://github.com/pjqwudi1/my_vuln/blob/main/totolink/vuln_24/24.md



用grep找文件，锁定两个文件，分别审计

```
riria@riria-virtual-machine: ~/桌面/_A3100R.web.extracted
grep: 不适用的选项 -- N
用法: grep [选项]... 模式 [文件]...
尝试使用 'grep --help' 来获得更多信息。
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$ grep -RNL "setUploadSetting"
grep: 不适用的选项 -- N
用法: grep [选项]... 模式 [文件]...
尝试使用 'grep --help' 来获得更多信息。
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$ grep -Rnl "setUploadSetting"
squashfs-root/lib/cste_modules/upgrade.so
squashfs-root/web_cste/cgi-bin/cstecgi.cgi
squashfs-root-0/lib/cste_modules/upgrade.so
squashfs-root-0/web_cste/cgi-bin/cstecgi.cgi
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$ grep -Rnl "FileName"
squashfs-root/bin/tcpdump
squashfs-root/bin/forceupg
squashfs-root/lib/cste_modules/upgrade.so
squashfs-root/web_cste/cgi-bin/cstecgi.cgi
squashfs-root-0/bin/tcpdump
squashfs-root-0/bin/forceupg
squashfs-root-0/lib/cste_modules/upgrade.so
squashfs-root-0/web_cste/cgi-bin/cstecgi.cgi
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$
```

ida查字符串，确认函数位于upgrade.so里边



开始审计，但信息也说明了漏洞在FileName字段附近，所以找FileName相关的。

```
int __fastcall setUploadSetting(int a1, int a2, int a3)
{
    const char *Var; // $s4
    const char *v7; // $s3
    int Object; // $s1
    int String; // $v0
    signed int v10; // $s0
    int v11; // $v0
    int v12; // $v0
    char *v13; // $s2
    FILE *v14; // $v0
    FILE *v15; // $s4
    int head_offset; // $s3
    int v17; // $v0
    int v18; // $v0
    int v19; // $v0
```

```

int v20; // $v0
int v21; // $v0
int v22; // $v0
int v23; // $v0
void *v24; // $s0
void *v25; // $s0
int v27; // [sp+18h] [-130h] BYREF
int v28; // [sp+1Ch] [-12Ch] BYREF
char v29[128]; // [sp+20h] [-128h] BYREF
int v30[4]; // [sp+A0h] [-A8h] BYREF
__int16 v31; // [sp+B0h] [-98h]
char v32[128]; // [sp+B4h] [-94h] BYREF
int v33[5]; // [sp+134h] [-14h] BYREF

v27 = 0;
v28 = 0;
memset(v29, 0, sizeof(v29));
var = (const char *)websGetVar(a2, "FileName", ""); // 可控变量
v7 = (const char *)websGetVar(a2, "ContentLength", "");
memset(v30, 0, sizeof(v30));
v31 = 0;
memset(v32, 0, sizeof(v32));
memset(v33, 0, 16);
Object = cJSON_CreateObject();
apmib_get(7102, v30);
sprintf(v32, "cat %s | grep %s", var, (const char *)v30); // 一条命令，传参给v32.实际命令为cat var | grep v30
if ( getCmdStr(v32, v33, 16) == -1 ) // getCmdStr，如何确定这个函数调用了getCmdStr?
{
    String = cJSON_CreateString("MM_ConfigFileInvalid");
    cJSON_AddItemToObject(Object, "settingERR", String);
LABEL_21:
    v25 = (void *)cJSON_Print(Object);
    websGetCfgResponse(a1, a3, v25);
    free(v25);
    cJSON_Delete(Object);
    return 0;
}

```

看到可控变量Var进入了getCmdStr这个非常像系统调用的函数，查[blog](#)也可发现这就是一个调用了system的函数。但对于分析器来说，如何确认这个就是系统调用函数呢？

```
riria@riria-virtual-machine: ~/桌面/_A3100R.web.extracted
squashfs-root-0/web_cste/js/language_en.js:var MSG_Wds="You could set Wireless Distribution System (WDS).";
squashfs-root-0/web_cste/js/language_pt.js:var MSG_Wds ="Você pode definir Wireless Distribution System (WDS) .";
squashfs-root-0/web_cste/adm/dos.asp: 'Whole System Flood: SYN&sysfloodSyn',
squashfs-root-0/web_cste/adm/dos.asp: 'Whole System Flood: FIN&sysfloodFin',
squashfs-root-0/web_cste/adm/dos.asp: 'Whole System Flood: UDP&sysfloodUdp',
squashfs-root-0/web_cste/adm/dos.asp: 'Whole System Flood: ICMP&sysfloodIcmp',
匹配到二进制文件 squashfs-root-0/web_cste/cgi-bin/upload_settings.cgi
匹配到二进制文件 squashfs-root-0/web_cste/cgi-bin/cstecgi.cgi
匹配到二进制文件 squashfs-root-0/web_cste/cgi-bin/downloadFile.cgi
匹配到二进制文件 squashfs-root-0/web_cste/cgi-bin/upload_bootloader.cgi
匹配到二进制文件 squashfs-root-0/web_cste/cgi-bin/upload.cgi
匹配到二进制文件 4C.jffs2
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$ grep -irw "getCmdStr"
匹配到二进制文件 squashfs-root-0/lib/cste_modules/global.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/lan.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/upgrade.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/wireless.so
匹配到二进制文件 squashfs-root-0/lib/libcstelib.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/global.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/lan.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/upgrade.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/wireless.so
匹配到二进制文件 squashfs-root-0/lib/libcstelib.so
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$ grep -ir "getCmdStr"
匹配到二进制文件 squashfs-root-0/lib/cste_modules/global.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/lan.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/upgrade.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/wireless.so
匹配到二进制文件 squashfs-root-0/lib/libcstelib.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/global.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/lan.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/upgrade.so
匹配到二进制文件 squashfs-root-0/lib/cste_modules/wireless.so
匹配到二进制文件 squashfs-root-0/lib/libcstelib.so
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted$
```

运气好，在/lib/libcstelib.so里找到了getCmd的调用

```
int __fastcall getCmdStr(int a1, _BYTE *a2, int a3)
{
    //a1可控
    int v5; // $v0
    int v6; // $s1
    int v7; // $s0
    _BYTE *v8; // $v0

    v5 = popen(a1, "r");//v5=popen("cat var | grep v30", "r");
    v6 = v5;
    v7 = -1;
    if ( v5 )
    {
        if ( fgets(a2, a3, v5) )
        {
            v8 = (_BYTE *)strchr(a2, 10);
            v7 = 0;
            if ( v8 )
                *v8 = 0;
        }
        else
        {
            *a2 = 0;
            v7 = -1;
        }
        pclose(v6);
    }
    return v7;
}
```

发现变量进入了popen()里边，从Linux函数里边可以看到popen定义：<https://www.cnblogs.com/52p hp/p/5722238.html>

popen()会调用fork()产生子进程，然后从子进程中调用/bin/sh -c来执行参数command的指令。参数type可使用“r”代表读取，“w”代表写入。依照此type值，popen()会建立管道连到子进程的标准输出设备或标准输入设备，然后返回一个文件指针。随后进程便可利用此文件指针来读取子进程的输出设备或是写入到子进程的标准输入设备中。此外，所有使用文件指针(FILE*)操作的函数也都可以使用，除了fclose()以外。

如果 `type` 为 `r`，那么调用进程读进 `command` 的标准输出。
如果 `type` 为 `w`，那么调用进程写到 `command` 的标准输入。

示例：

```
#include<stdio.h>

main()
{
    FILE *fp;
    char buffer[80];
    fp = popen("cat /etc/passwd", "r");
    fgets(buffer, sizeof(buffer), fp);
    printf("%s", buffer);
    pclose(fp);
}
```

显然，是命令执行没错了，所以遇到这种已经定义好的函数，需要准备一个数据库来识别

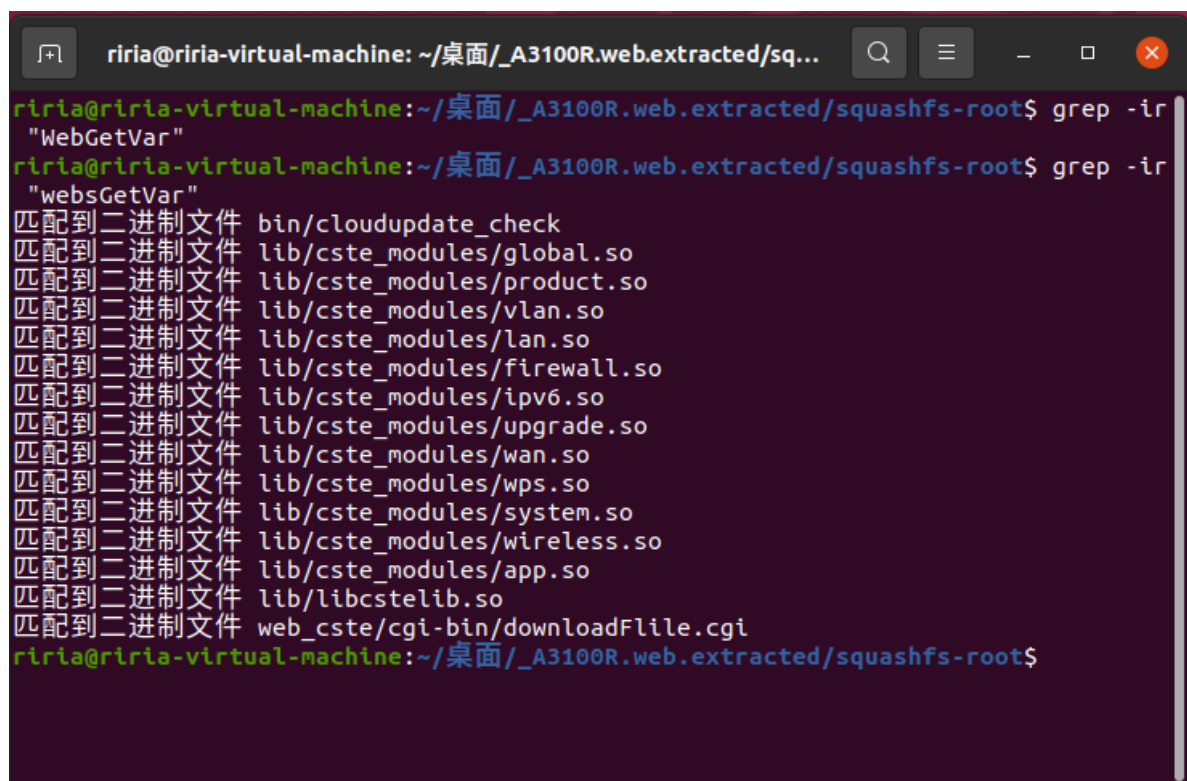
仿真复现

难点在于构造exp以及让特定代码运行起来

对于`cat Var | grep v30`的形式，需要把后边注释掉，然后让前边`cat`正常执行再执行我们想要的指令。

所以现在我们需要在实际仿真中知道Var原本要填入什么值。

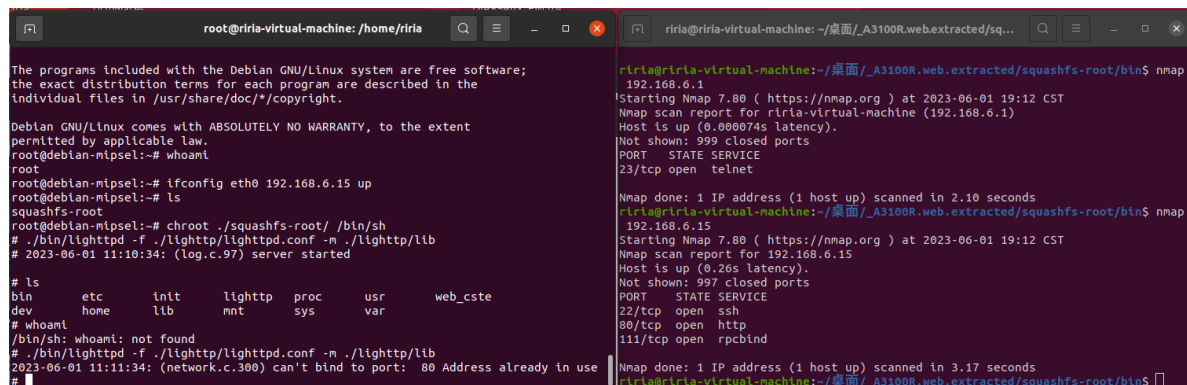
首先得知道获取Var的这个函数，WebGetVar是怎么工作的：



```
riria@riria-virtual-machine: ~/桌面/_A3100R.web.extracted/sq...
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted/squashfs-root$ grep -ir
"WebGetVar"
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted/squashfs-root$ grep -ir
"websGetVar"
匹配到二进制文件 bin/cloudupdate_check
匹配到二进制文件 lib/cste_modules/global.so
匹配到二进制文件 lib/cste_modules/product.so
匹配到二进制文件 lib/cste_modules/vlan.so
匹配到二进制文件 lib/cste_modules/lan.so
匹配到二进制文件 lib/cste_modules/firewall.so
匹配到二进制文件 lib/cste_modules/ipv6.so
匹配到二进制文件 lib/cste_modules/upgrade.so
匹配到二进制文件 lib/cste_modules/wan.so
匹配到二进制文件 lib/cste_modules/wps.so
匹配到二进制文件 lib/cste_modules/system.so
匹配到二进制文件 lib/cste_modules/wireless.so
匹配到二进制文件 lib/cste_modules/app.so
匹配到二进制文件 lib/libcstelib.so
匹配到二进制文件 web_cste/cgi-bin/downloadFile.cgi
riria@riria-virtual-machine:~/桌面/_A3100R.web.extracted/squashfs-root$
```

东西很多，但第一个盯上的是`downloadFile.cgi`文件，进去看看。然后发现不是，之后在`cloudupdate_check`里找到了`WebGetVar`的定义，但不知道是不是IAT的问题，看到的是`tunk`的形式。

qemu仿真启动



The image shows two terminal windows. The left window is a root shell on a virtual machine named 'riria'. It shows the installation of 'squashfs-root' and the configuration of 'lighttpd' to listen on port 80. The right window shows an nmap scan of the virtual machine's IP address (192.168.6.1). The scan results show that port 23/tcp is open (telnet) and port 80/tcp is open (http). The scan also shows that 997 ports are closed.

但是web端连接不了，不知道为什么。不过也能看到它开启的几个服务了。后边又试了试，能跑了，好吧，应该是之前网络配置不对

登陆界面的post包：

```
POST /cgi-bin/cstecgi.cgi?action=login&flag=1 HTTP/1.1

Host: 192.168.6.15

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/113.0

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

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

Content-Length: 20

Origin: http://192.168.6.15

Connection: close

Referer: http://192.168.6.15/mobile/login.asp

Upgrade-Insecure-Requests: 1

username=123&password=123
```

涉及了cstecgi.cgi这个文件，而且第一个路径是/cgi-bin，说明web根目录在web_cste。构造包，但是进不去路由界面，只能发包测试能不能到那个函数。

然后顺着login.asp里边的参数找函数，在cstecgi.cgi文件中有下方代码：

```

}
else if ( v28 && strstr(v28, "action=login") )
{
    v15 = strstr(v28, "flag=1") != 0;
    v24 = (const char *)getenv("http_host");
    memset(v33, 0, sizeof(v33));
    if ( v15 )
        sprintf(
            v33,
            "{\"topicurl\":\"setting/loginAuth\",\"loginAuthUrl\":\"%s&http_host=%s&flag=1\"}",
            (const char *)v34,
            v24);
    else
        sprintf(v33, "{\"topicurl\":\"setting/loginAuth\",\"loginAuthUrl\":\"%s&http_host=%s\"}", (const char *)v34, v24);
    v29 = (int *)v33;
    v3 = cJSON_Parse(v33);
}
else if ( v28 && strstr(v28, "action=upload&upgrade") )
{
    v20 = (const char *)getenv("UPLOAD_FILENAME");
    v14 = strstr(v28, "flag=1") != 0;
    memset(v33, 0, sizeof(v33));
    sprintf(
        v33,
        "{\"topicurl\":\"setting/setUpgradeFW\",\"FileName\":\"%s\",\"Flags\":\"%d\",\"ContentLength\":\"%s\"}",
        v20,
        v14,
        v26);
    v29 = (int *)v33;
    v3 = cJSON_Parse(v33);
}
}

```

在action=login下方不远处，就算action=upload的动作，涉及到了setUpgradeFW这个函数以及FileName参数，好家伙，直接到了[CVE-2022-26210](#)的漏洞点了。setUploadSetting的参数在下边一点：

```

}
else if ( v28 && strstr(v28, "action=upload&upgrade") )
{
    v20 = (const char *)getenv("UPLOAD_FILENAME");
    v14 = strstr(v28, "flag=1") != 0;
    memset(v33, 0, sizeof(v33));
    sprintf(
        v33,
        "{\"topicurl\":\"setting/setUpgradeFW\",\"FileName\":\"%s\",\"Flags\":\"%d\",\"ContentLength\":\"%s\"}",
        v20,
        v14,
        v26);
    v29 = (int *)v33;
    v3 = cJSON_Parse(v33);
}
else if ( v28 && strstr(v28, "action=save&setting") )
{
    v25 = (const char *)getenv("http_host");
    memset(v33, 0, sizeof(v33));
    sprintf(v33, "{\"topicurl\":\"setting/getSaveConfig\",\"http_host\":\"%s\"}", v25);
    v29 = (int *)v33;
    v3 = cJSON_Parse(v33);
}
else if ( v28 && strstr(v28, "action=upload&setting") )
{
    v21 = (const char *)getenv("UPLOAD_FILENAME");
    memset(v33, 0, sizeof(v33));
    sprintf(v33, "{\"topicurl\":\"setting/setUploadSetting\",\"FileName\":\"%s\",\"ContentLength\":\"%s\"}", v21, v26);
    v29 = (int *)v33;
    v3 = cJSON_Parse(v33);
}
else

```

最后在验证参数时卡在了模拟上，虽然http服务开启了但是没有返回，应该是缺东西。不过这个洞涉及的固件挺多的，可以换个固件再模拟模拟。