路由器无限重启救砖之旅

挖掘路由器漏洞时，发现了一个命令执行漏洞，正好有重启设备的需求，手残通过漏洞点传递 reboot 命令重启，于是设备变砖了。

前言

挖掘路由器漏洞时，发现了一个命令执行漏洞，正好有重启设备的需求，手残通过漏洞点传递 reboot 命令重启，于是设备变砖了。

分析变砖原因，发现传递的参数会保存 NVRAM 中，开机的时候会读取并执行，于是造成了设备的拒绝服务。手头这个设备也是好不容易淘来的，这可怎么是好。开始救砖吧！

尝试一——恢复记忆

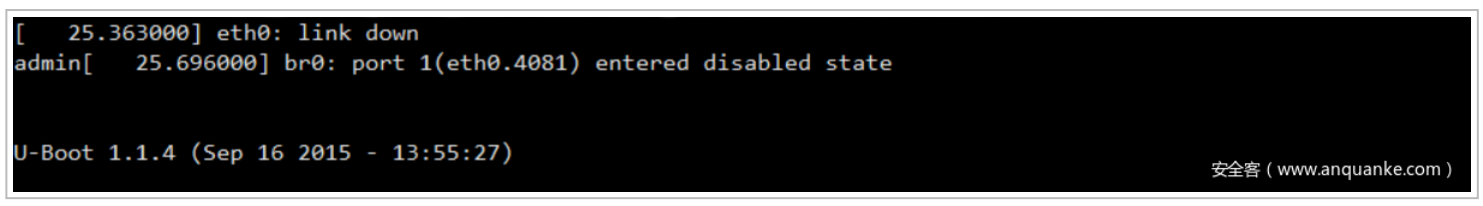
由于 reboot 命令存储到了路由器的 NVRAM 配置中，将路由器**恢复出厂设置**还原 NVRAM 中的值理论上可行。正好路由器上有 reset 按键，“摁”住它一段时间，测试了好几次没反应。这种方法就不回来。猜测是负责恢复出厂设置程序在设备自动重启之间没有启动。

尝试二——争分夺秒

软的不行来硬的，直接拆。拆了发现预留有串口。通过串口进入系统 shell，与自动重启争分夺秒。希望 web 服务启动能有点延时，在重启之前，删除 reboot。

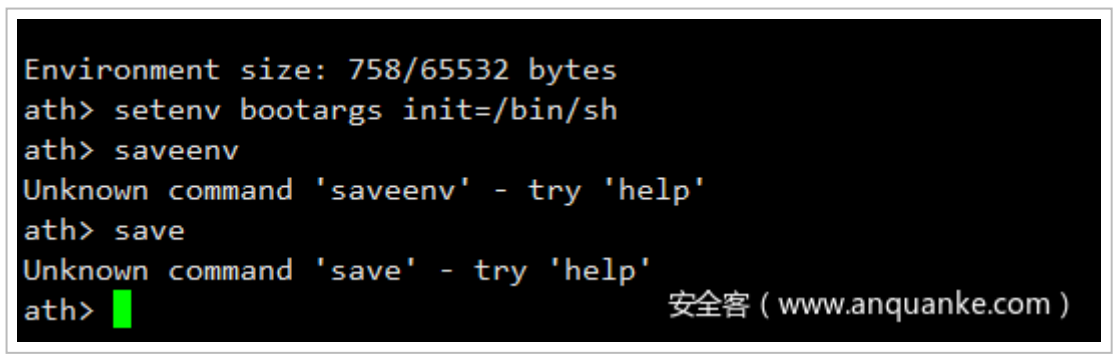
```
nvram set name=""
```

输入了用户名密码，还没进入 shell 就重启了。这个方法也不行，时间来不及。



尝试三——紧急救援

进入**单用户模式**，修改启动参数进入单用户模式，U–Boot 没有 saveenv 命令，使用 setenv 修改启动参数后无法保存。这种方法也受阻了，如果能进入单用户模式，可以恢复参数进行救砖。



尝试四——内存修改

首先提取出固件定位到问题所在，然后修改好固件后，最后在 U–Boot 中直接刷写。

1. 提取固件

读取内存：Flash 在内存的地址使用命令 bdinfo 查看，。

```
ath> bdinfo
boot_params = 0x87F77FB0
memstart    = 0x80000000
memsize     = 0x08000000
flashstart  = 0x9F000000
flashsize   = 0x01000000
flashoffset = 0x0002BD20
ethaddr     = 00:AA:BB:CC:DD:EE
```

```
ip_addr    = 10.10.10.123
baudrate   = 115200 bps
```

Flash 的起始地址为 0x9F000000，大小为 0x01000000（16M）。16M = 16777216 Bit，16777216/32 = 524288=0x80000。启动 xshell 的日志记录功能。等待两个小时左右，完成读取。

```
ath> md 0x9F000000 80000
9f000000: 100000ff 00000000 100000fd 00000000    ................
9f000010: 10000175 00000000 10000173 00000000    ...u.......s....
9f000020: 10000171 00000000 1000016f 00000000    ...q.......o....
9f000030: 1000016d 00000000 1000016b 00000000    ...m.......k....
9f000040: 10000169 00000000 10000167 00000000    ...i.......g....
```

然后把日志中的内容转化为二进制。

```
import re

pre_index = 0
index = 0
#检查日志中的内容是否完整
with open(r"md_flash.log".encode(),encoding="utf-8") as log:
    for line in log.readlines():
        index = int(line.split(":")[0],16)
        if pre_index == 0:
            pre_index = index
            continue
        if index - pre_index != 0x10:
            print(hex(pre_index)," to ",hex(index),"data absence")
            break
        else:
            pre_index = index

with open(r"md_flash.log".encode(),encoding="utf-8") as log:
    data = log.read()

#末尾加一个换行符，方便正常统一处理
data = data+"\n"

# 去掉字符
data = re.sub(r"\s\s\s\s.*\n","",data)
# 去掉地址
data = re.sub(r"[0-9a-zA-Z]{8,8}:\s","",data)
# 去掉空格
data = re.sub(r"\s","",data)

with open("flash.bin","bw") as f:
    f.write(bytes.fromhex(data))
```

2. 分析固件

binwalk 分析对固件的分析结果。

```
root@[redacted]:~# binwalk [redacted].bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
145488        0x23850         U-Boot version string, "U-Boot 1.1.4 (Sep 16 2015 - 13:55:27)"
145728        0x23940         CRC32 polynomial table, big endian
327680        0x50000         uImage header, header size: 64 bytes, header CRC: 0xF5B38ECB, created: 2019-07-19 09:31:05, image size: 896422
4 bytes, Data Address: 0x80060000, Entry Point: 0x803A3B00, data CRC: 0x598FC926, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compres
sion type: lzma, image name: "Linux Kernel Image"
327744        0x50040         LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 11906908 bytes
14483468      0xDD000C        gzip compressed data, maximum compression, from Unix, last modified: 2020-11-25 18:56:39
16384128      0xFA0080        bzip2 compressed data, block size = 900k
16646156      0xFE000C        gzip compressed data, maximum compression, from Unix, last modified: 2020-11-25 18:56:39  安全客 ( www.anquanke.com )
```

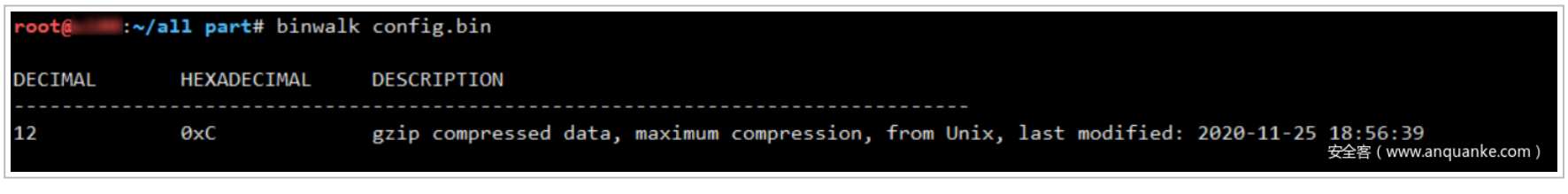
现在手里有串口，当然得参考串口日志信息。根据串口打印的 Flash 中配置分区的信息。

```
[    9.465000] Creating 9 MTD partitions on "spi0.0":
[    9.519000] 0x000000fe0000-0x000000ff0000 : "Config"
```

使用 dd 拆分出各模块，下面是可能与 NVRAM 有关的模块。

```
root@kali#dd if=flash.bin of=config.bin bs=1 count=65535 skip=16646144
65535+0 records in
65535+0 records out
65535 bytes (66 kB, 64 KiB) copied, 1.46555 s, 44.7 kB/s
```

binwalk 分析出 config 中的内容是经过 gzip 压缩的。



binwalk 提取出配置文件，里面是明文存储着的配置数据。

找到了罪魁祸首。

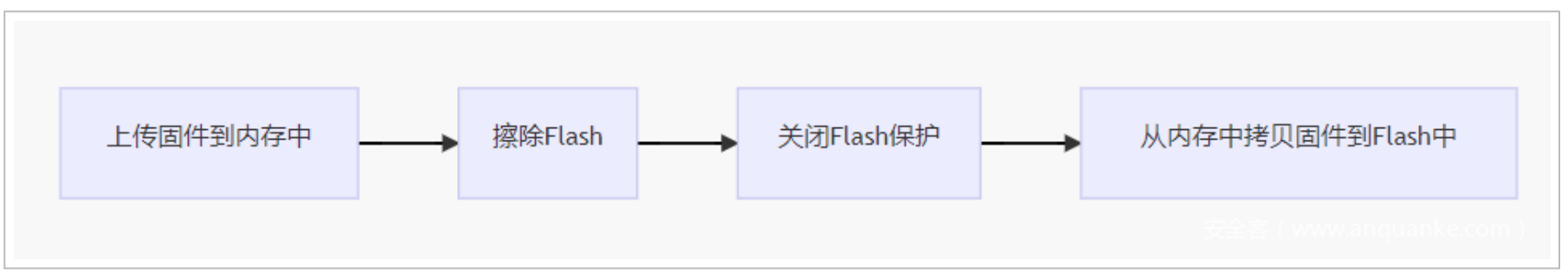
修改好之后，进行压缩。

```
root@kali:~# gzip C  -9 -c > config_m.bin
```

最后加上加上分区头，就重构了配置分区，接下来就要想办法写入到 Flash 中。

3. 修改 Flash

搞 IOT 安全的并不是都熟悉嵌入式开发，搞嵌入式开发的不一定理解我们的需求，问了搞嵌入式的朋友他会不会。我是第一次通过 U-Boot 刷写固件，由于缺乏基础知识走了不少弯路，写这篇文章的目的也是帮助有同样需求的人少走弯路。U-Boot 固件刷写不能直接写入到 Flash 中，需要按照下图的流程操作。



首先尝试用 FTP、loady、loads 上传文件，实际使用时选择一种即可。最后重新刷写配置分区，救活路由器。

1.1) 使用 FTP 上传固件

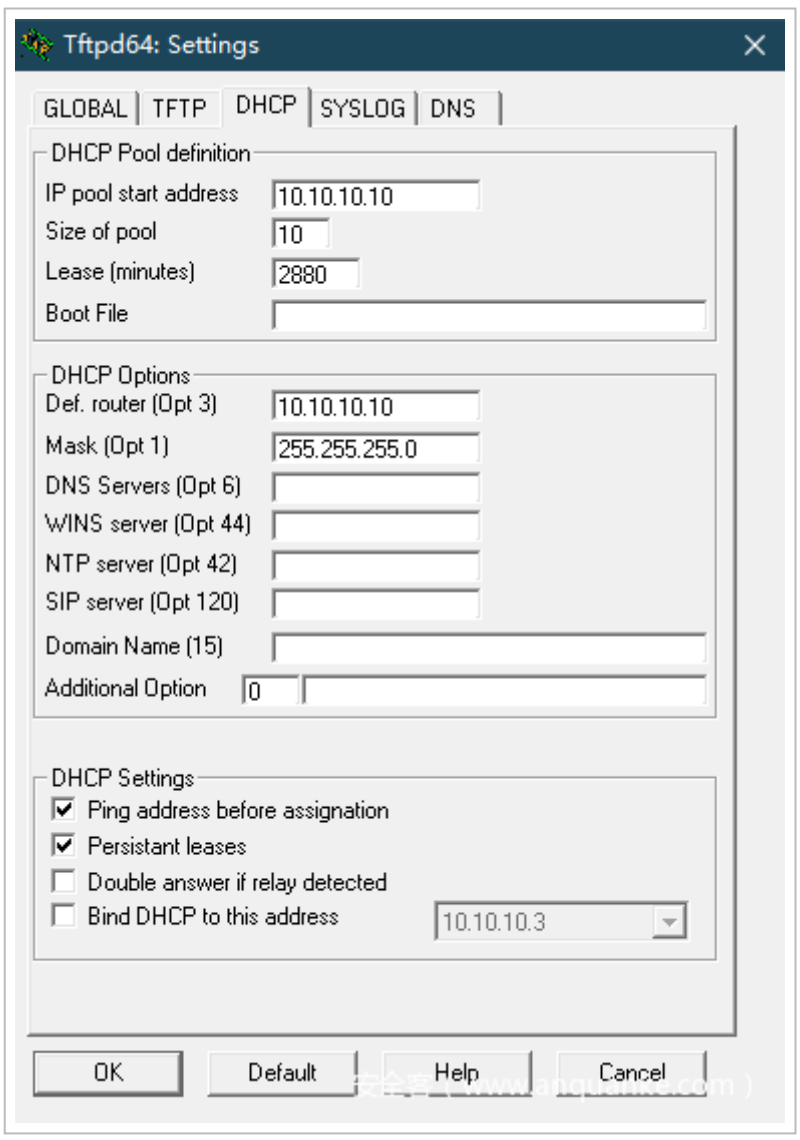
printenv 命令查看环境变量可知预设服务器的地址为 10.10.10.3。

```
ath> base
Base Address: 0x00000000
ath> printenv
bootargs=
bootcmd=bootm 0x9f050000
bootdelay=2
baudrate=115200
ethaddr=0x00:0xaa:0xbb:0xcc:0xdd:0xee
ipaddr=10.10.10.123
serverip=10.10.10.3
dir=
```


找一根网线连接路由器和电脑，然后给电脑设置静态 IP 地址 10.10.10.3，掩码 255.255.255.0。然后，测试一下连通性。在路由器串口中 ping 电脑。

```
ath> ping 10.10.10.3
Trying eth0
enet0 port4 up
dup 1 speed 100
Using eth0 device
host 10.10.10.3 is alive
```


host 10.10.10.3 存活说明连接成功。另外，还准备一个 TFTP 服务器，刚开始用的 FileZilla ，不行发现还需要 DHCP 服务器，有找个 dhcpsrv ，最后发现还是 TFTPD 比较好用。打开就能用，配置也很简单，FTP 设置一下路径，DHCP 需要设置 IP 等，如下图。

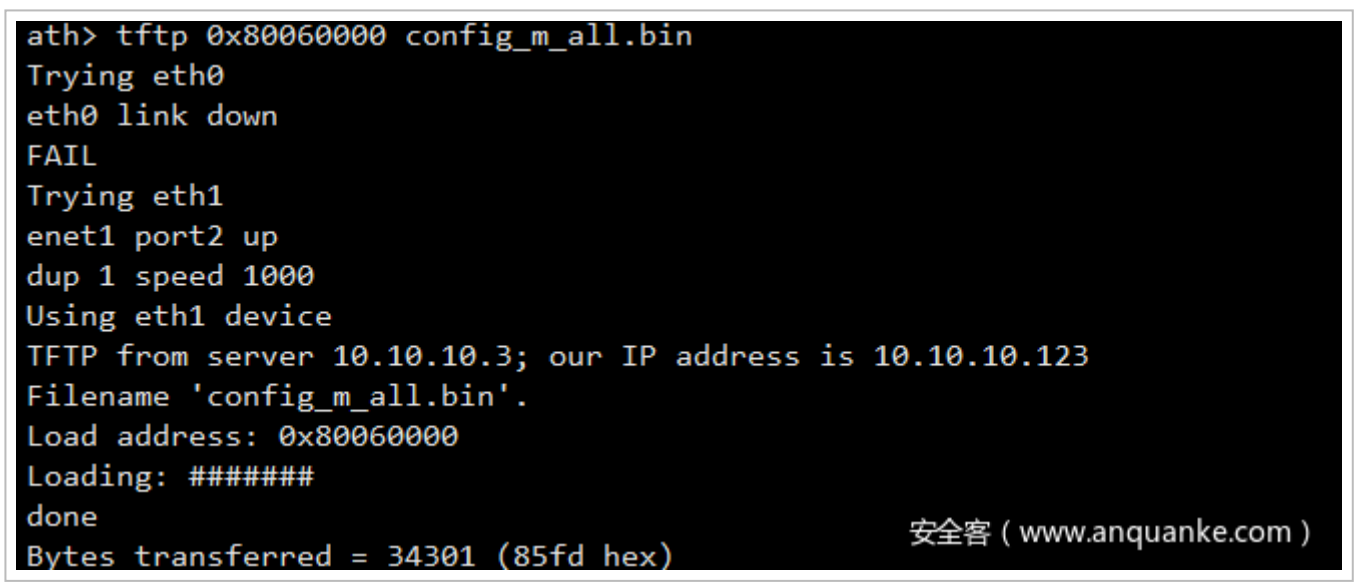


参考环境变量中的 ip，如果环境变量中没有可以参考的，那就需要在内存中寻找一块足够大的空白区域

参考环境变量中的 la，如果环境变量中没有可以参考的，那就需要在内存中寻找一块足够大的空白区域。

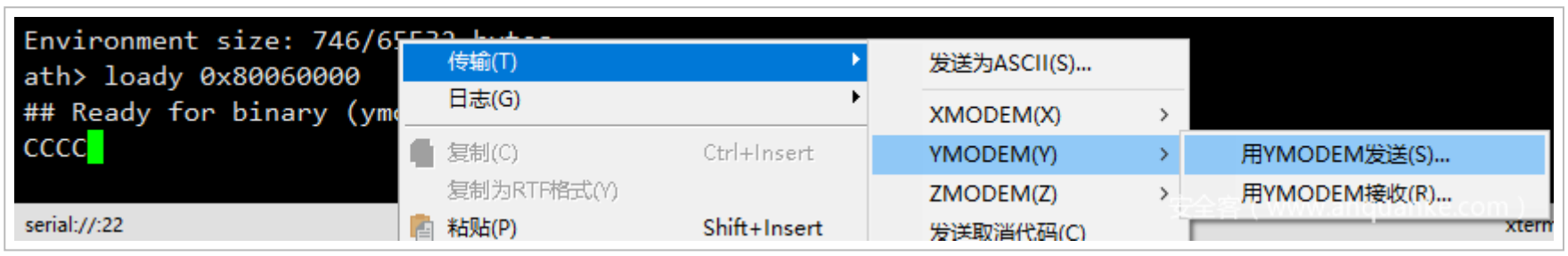
```
la=tftp 0x80060000 data.bin&&erase 0x9fff0000 +$filesize&&protect off all&&cp.b $fileaddr 0x9fff0000 $filesize
```

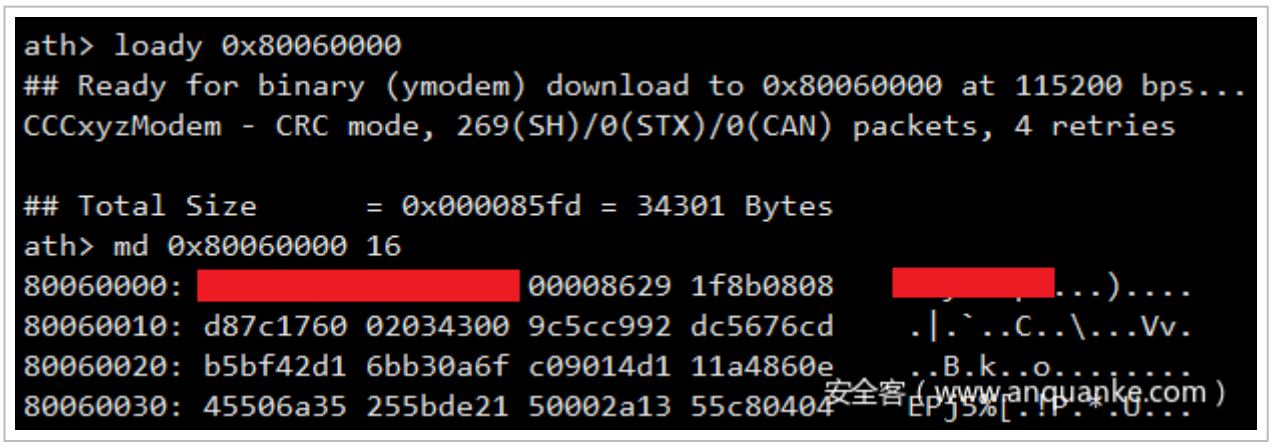
将固件加载到地址 0x80060000。



1.2) 使用 loady 上传固件

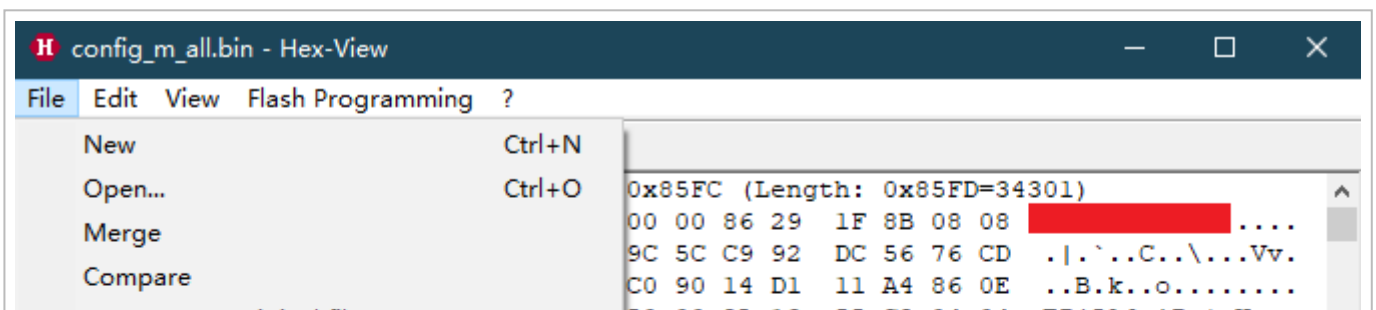
命令格式为 `loady address`，然后使用 YMODEM 发送修改后的文件。

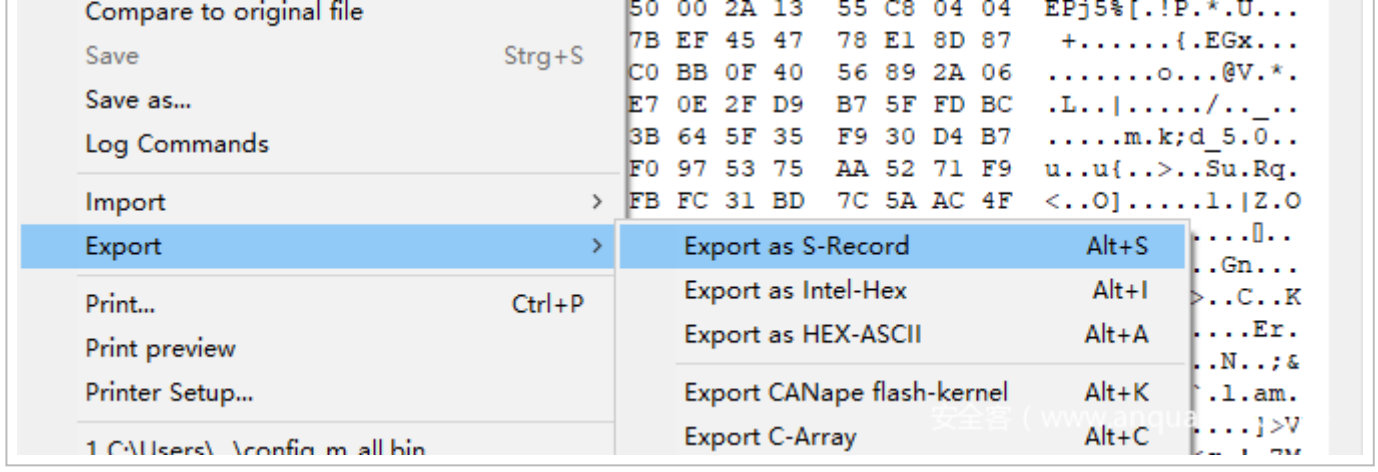




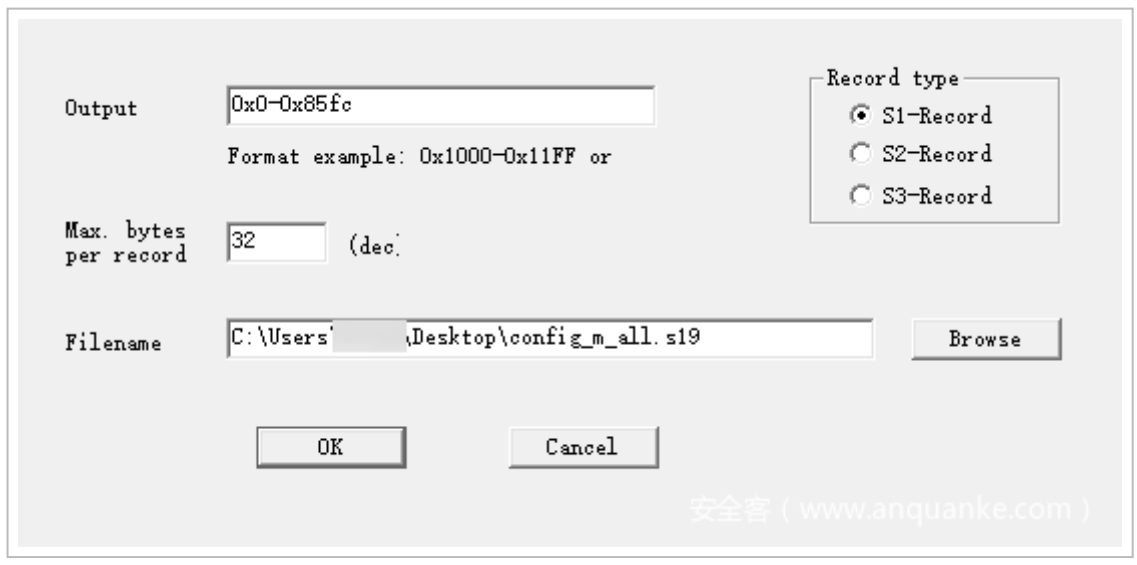
1.3) 使用 loads 上传固件

首先将二进制文件转为 S-Record 文件。





使用默认配置，导出即可。



S-Record 文件中的数据都是以 ASCII 码的格式存储，正好使用 Xshell ASCII 字符传输修改后的文件。命令格式为 `loads address` 。

选择文件，等待片刻写入完成。

```
ath> loads 0x80060000
## Ready for S-Record download ...
....
......
## First Load Addr = 0x80060000
## Last  Load Addr = 0x800685FC
## Total Size      = 0x000085FD = 34301 Bytes
## Start Addr      = 0x0000FFFF
```

2) 刷写配置分区

刷血配置分区，加载到内存的数据拷贝到 Flash 中。

1. 擦除 Config 分区

```
ath> erase 9ffe0000 9ffeffff
Erasing flash...
First 0xfe last 0xfe sector size 0x10000
Erased 1 sectors
```

```
ath> md 9ffe0000
9ffe0000: ffffffff ffffffff ffffffff ffffffff    ................
9ffe0010: ffffffff ffffffff ffffffff ffffffff    ................
9ffe0020: ffffffff ffffffff ffffffff ffffffff    ................
9ffe0030: ffffffff ffffffff ffffffff ffffffff    ................
9ffe0040: ffffffff ffffffff ffffffff ffffffff    ................
```


2. 关闭 flash 保护

```
ath> protect off all
Un-Protect Flash Bank # 1
```


3. 从 Momory 中复制到 Flash 中。

```
ath> cp.b 80060000 9ffe0000 85fd
Copy to Flash... write addr: 9ffe0000
done
```


4. 查看写入的数据

```
ath> md 9ffe0000
9ffe0000: [redacted] 1f8b0808 [redacted]
9ffe0010: d87c1760 02034300 9c5cc992 dc5676cd    .|.`..C..\...Vv.
9ffe0020: b5bf42d1 6bb30a6f c09014d1 11a4860e    ..B.k..o........
9ffe0030: 45506a35 255bde21 50002a13 55c80404    EPj5%[.!P.*.U...
9ffe0040: 202bab18 fca0fe02 7bef4547 78e18d87     +......{.EGx...
9ffe0050: f017f81b bcf5bd6f c0bb0f40 56892a06    .......o...@V.*.
```


4. 重启

输入 reset 重启，路由器正常运行，救砖成功。

尝试五——物理伤害

除了 U-Boot 刷写还可以直接**物理操作**，使用 FT232H 连接 SOP8 封装的 SPI Flash，用 Flashrom 刷入固件。

压缩修改后的配置文件。

```
root@kali:~# gzip C  -9 -c > config_m.bin
```

截取分区头。

```
root@kali:~# dd if=../config.bin of=header.bin bs=1 count=12
12+0 records in
12+0 records out
12 bytes copied, 0.000132176 s, 90.8 kB/s
```

计算需要填充的 `0xFF` 的大小，然后使用 dd 创建填充块。

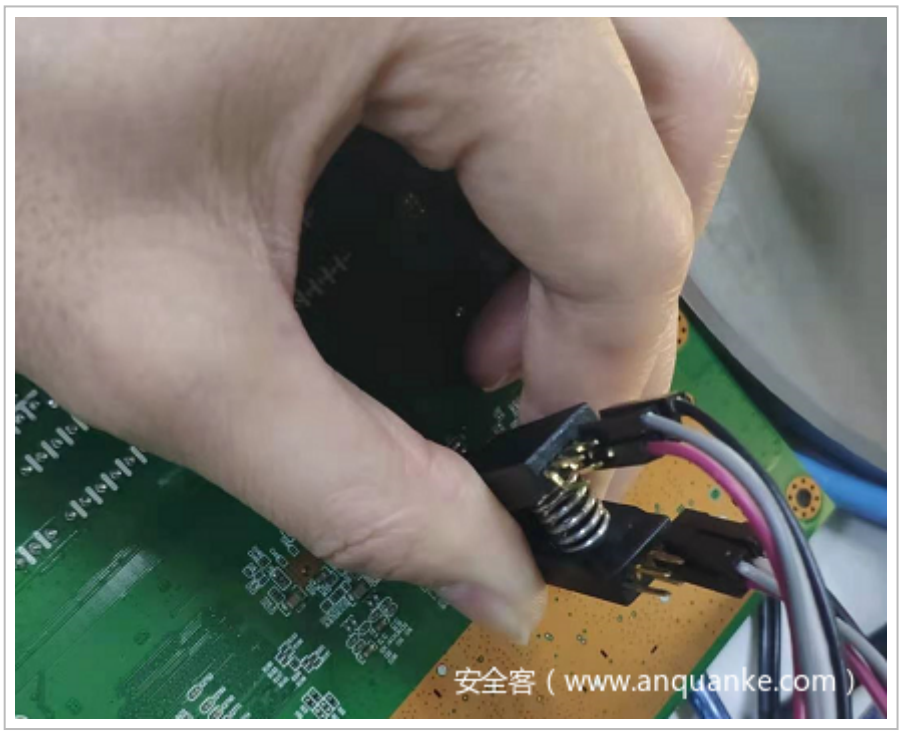
```
root@kali:~# ls -l config_m.bin
-rw-r--r-- 1 root root 34289 Jan 31 23:08 config_m.bin
root@kali:~# ls -l config.bin
-rwxrw-rw- 1 root root 65535 Jan 27 03:15 config.bin
root@kali:~# python3 -c "print(65535-12-34289)"
root@kali:~# dd if=/dev/zero bs=1 count=31234 | tr "\000" "\377" >ff.bin
31234+0 records in
31234+0 records out
31234 bytes (31 kB, 31 KiB) copied, 0.0801646 s, 390 kB/s
```

最后，将所有的分区拼接起来形成完整的固件。

```
root@kali:~# cat header.bin config_m.bin ff.bin  > config_new.bin
root@kali:~# ls -l config_new.bin
-rw-r--r-- 1 root root 65535 Jan 31 23:24 config_new.bin
```

以上操作完成了固件的重打包，然后就可以使用 Flashrom 写入新固件。

SOP 夹子夹不稳，只能手扶着。



写入时间还是比较长，手有点受不了，可能导致接触不良，写入失败。

```
root[redacted]:~/all part# flashrom -p ft2232_spi:type=232H -w header1.bin
flashrom v1.2 on Linux 5.3.0-kali2-amd64 (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Using clock_gettime for delay loops (clk_id: 1, resolution: 1ns).
Found Winbond flash chip "W25Q128.V" (16384 kB, SPI) on ft2232_spi.
Reading old flash chip contents... done.
Erasing and writing flash chip... FAILED at 0x00009000! Expected=0xff, Found=0x02, failed byte count from 0x00009000-0x00009fff: 0xfcf
ERASE FAILED!
Reading current flash chip contents... done. Looking for another erase function.
ftdi_read_data: -4, usb bulk read failed
get_buf failed: 1
ftdi_write_data: -1, usb bulk write failed
send_buf failed at end: 1
RDSR failed!
ftdi_write_data: -1, usb bulk write failed
send_buf failed at end: 1
spi_write_cmd failed during command execution at address 0xbf000
ftdi_write_data: -1, usb bulk write failed
send_buf failed before read: 1
ftdi_write_data: -1, usb bulk write failed
send_buf failed at end: 1
RDSR failed!
Reading current flash chip contents... ftdi_write_data: -1, usb bulk write failed
send_buf failed before read: 1
ftdi_write_data: -1, usb bulk write failed
send_buf failed at end: 1
Can't read anymore! Aborting.
FAILED!
Uh oh. Erase/write failed. Checking if anything has changed.
Reading current flash chip contents... ftdi_write_data: -1, usb bulk write failed
send_buf failed before read: 1
ftdi_write_data: -1, usb bulk write failed
send_buf failed at end: 1
Can't even read anymore!
Your flash chip is in an unknown state.
Please report this on IRC at chat.freenode.net (channel #flashrom) or
mail flashrom@flashrom.org, thanks!
```


然后，就使用芯片测试夹连接，这样稳一下。

此时写入成功，但是验证失败。

```
root@      ~/all part# flashrom -p ft2232_spi:type=232H -w flash_new.bin
flashrom v1.2 on Linux 5.3.0-kali2-amd64 (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Using clock_gettime for delay loops (clk_id: 1, resolution: 1ns).
Found Winbond flash chip "W25Q128.V" (16384 kB, SPI) on ft2232_spi.
Reading old flash chip contents... done.
Erasing and writing flash chip... Erase/write done.
Verifying flash... FAILED at 0x000003f3! Expected=0x7d, Found=0x3d, failed byte count from 0x00000000-0x00ffffff: 0x216b13
Your flash chip is in an unknown state.
Please report this on IRC at chat.freenode.net (channel #flashrom) or
mail flashrom@flashrom.org, thanks!
```


先不管上电，并双手合十，登录界面维持住了，Web 管理端也能访问了，救砖成功。

其他方法

除以上 5 种方法，还有更简单粗暴的方法，但需要一个正常的设备，或之前备份过的正常且完整 Flash。操作比较简单直接读取正常设备的 Flash 并写入变砖的设备中即可。

总结

救砖，首先需要分析搬砖的原因，然后对症下药。我这次是配置参数引起的，所以我首先想到的是恢复出厂设置，但恢复出厂模式需要长按数秒，此时设备已经重启，这种方法就不行了；然后想到进入 shell 快速还原配置参数修复，结果是还没进入 shell，设备就重启了；设备重启是 Web 服务启动时触发，只要不启动 Web 服务设备就不会重启，于是就尝试了一下进入单用户模式，然而 U-Boot 中修改后的启动参数却无法保存； 从系统层的修复已经无计可施只能从底层出发，于是对固件进行提取分析，找到问题点修复后重新刷写，刷写尝试了不同的方法，一是通过 U-Boot 刷写，另外就是直接操作 Flash。

此次救砖还是花了不少时间的，通过实践了解了 NVRAM 的物理存储方式，还尝试了 U-Boot 中多种固件刷写方法。

参考

- U-Boot Quick Reference
- 抢救变砖的某款智能音箱