Tencent

# Rooting Every Android : From extension to exploitation

Di Shen a.k.a. Retme (@returnsme), James Fang (@idl3r)

Keen Lab of Tencent


black hat
EUROPE 2016

# About us

- Part of Keen Lab
- Interested in Android kernel security
  - Mostly the offensive part
- Responsible for many PHAs (non-malicious rooting)
  - PingPong root (CVE-2015-3636)
  - 1st public CVE-2015-1805 PoC (Dec 2015)
  - Multiple device specific root

Tencent

KEEN security lab

# Agenda

- Overview
  - Wi-Fi chipsets for Android
  - WEXT Attack Surface Analysis
  - Use device specific vulnerabilities to root them all
- Case Studies
  - Stack overflow vulnerability in Qualcomm WEXT
  - Data section overflow vulnerability in MTK WEXT
  - Use-After-Free vulnerability in Broadcom WEXT
- Google’s latest mitigation
- Conclusion

# Wi-Fi chipsets for Android

- Still Linux underneath
- Wireless Extension
  - Designed by Jean Tourrilhes in 1997
  - *"… a wireless API which would allow the user to manipulate any wireless networking device in a **standard** and **uniform** way"*
  - Implemented by all major wireless solution vendors
  - ***Will*** be replaced by cfg80211 with backward compatibility
    - Doesn't mean cfg80211 has ***fewer*** bugs

# WEXT Attack Surface Analysis

- Wireless Extension interfaces
  - Procfs node: /proc/net/wireless
    - Mostly a status query interface

```
root@xxx:/proc/net # cat wireless
Inter-| sta-|   Quality        |   Discarded packets               | Missed | WE
 face | tus | link level noise |  nwid  crypt   frag  retry   misc | beacon | 22
 wlan0: 0000    0     0     0        0      0      0      0      0        0
  p2p0: 0000    0     0     0        0      0      0      0      0        0
```

  - Everyone's favorite ioctl
    - Set/get configuration parameters
    - Issue commands

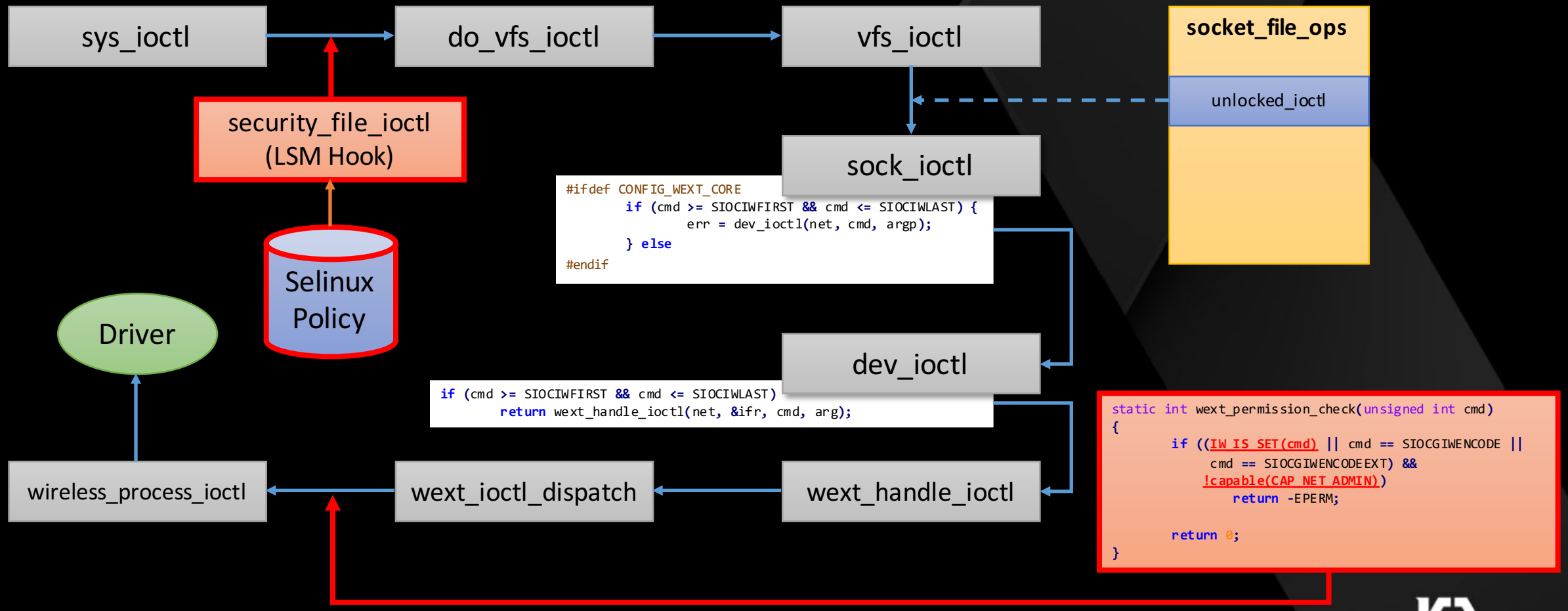
# WEXT Attack Surface Analysis (cont.)

- Ioctl can be issued on socket file descriptors
  - Ioctl command range: SIOCIWFIRST – SIOCIWLAST
    - Typical range 0x8B00 ~ 0x8BFF
  - Odd (quote) rule for get/set commands

```
/* Odd : get (world access), even : set (root access) */
#define IW_IS_SET(cmd)  (!((cmd) & 0x1))
#define IW_IS_GET(cmd)  ((cmd) & 0x1)
```

  - Merely any other access control
    - Before Google starting to take action
- "*These wireless extensions are not magic : each driver has to provide support for them...*" ——/include/uapi/linux/wireless.h

# WEXT Attack Surface Analysis (cont.)



# Root Them ALL…

- Broadcom
- Qualcomm
- Mediatek

Tencent

KEEN security lab

# Agenda

- Overview
  - Wi-Fi chipsets for Android
  - WEXT Attack Surface Analysis
  - Use device specific vulnerabilities to root them all
- Case Studies
  - **Stack overflow vulnerability in Qualcomm WEXT**
  - Data section overflow vulnerability in MTK WEXT
  - Use-After-Free vulnerability in Broadcom WEXT
- Google’s latest mitigation
- Conclusion

Tencent

KEEN security lab

# Case study 1 – CVE-2015-0570

- Reported by Renjia Lu ( 路人甲, aka. anonymous )
- CVE-2015-0570
- Advisory: https://www.codeaurora.org/projects/security-advisories/multiple-issues-wlan-driver-allow-local-privilege-escalation-cve-2015
  - Also fixed bugs caused by wrong cmd id (“odd” for set)

```
  int *value = (int *)extra;

+ if (!capable(CAP_NET_ADMIN)) {
+     VOS_TRACE(VOS_MODULE_ID_HDD, VOS_TRACE_LEVEL_ERROR,
+               FL("permission check failed"));
+     return -EPERM;
+ }
+
  if (wlan_hdd_validate_operation_channel(adapter, value[0]) !=
```

# Wait a Minute!

- Don’t we have stack protection?
- -fstack-protector is not as good as we thought
  - -fstack-protector-strong vs. -fstack-protector vs. **-fstack-protector-all**
  - -fstack-protector protects only ~2% of the functions
  - Overhead of -fstack-protector-all is too high for kernel
  - -fstack-protector-strong was recommended by Qualcomm and Google after this incident
    - http://android-developers.blogspot.com/2016/07/protecting-android-with-more-linux.html
- Requires GCC 4.9+
  - https://gcc.gnu.org/ml/gcc-patches/2012-06/msg00974.html

Tencent

KEEN security lab

# The Vulnerability

- Structure not triggering stack protection
- Stack overflow in function **wlan_hdd_set_filter**
  - Following data copy loop didn't check data length

```
int wlan_hdd_set_filter(hdd_context_t *pHddCtx, tpPacketFilterCfg pRequest, tANI_U8 sessionId)
{
    tSirRcvPktFilterCfgType    packetFilterSetReq = {0};
    ...

    switch (pRequest->filterAction)
    {
        case HDD_RCV_FILTER_SET:
            ...
            for (i=0; i < pRequest->numParams; i++)
            {
                ...
                packetFilterSetReq.paramsData[i].dataLength = pRequest->paramsData[i].dataLength;
                ...
                memcpy(&packetFilterSetReq.paramsData[i].compareData,
                        pRequest->paramsData[i].compareData, pRequest->paramsData[i].dataLength);
                memcpy(&packetFilterSetReq.paramsData[i].dataMask,
                        pRequest->paramsData[i].dataMask, pRequest->paramsData[i].dataLength);
                ...
            }
        ...
    }
    return 0;
}
```

```
struct PacketFilterParamsCfg
{
    uint8_t             protocolLayer;
    uint8_t             cmpFlag;
    uint8_t             dataOffset;
    uint8_t             dataLength;
    uint8_t             compareData[8];
    uint8_t             dataMask[8];
}

typedef struct
{
    uint8_t           filterAction;
    uint8_t           filterId;
    uint8_t           numParams;
    struct PacketFilterParamsCfg paramsData [5];
}tPacketFilterCfg, *tpPacketFilterCfg;
```

# How to Exploit

- Data flow is very straightforward
  - Fully controllable from user space
  - ioctl arg -> ifr (dev_ioctl) -> iwr (wireless_process_ioctl) -> tpPacketFilterCfg
- Old school stack overflow exploit
  - Giving calculated data length
  - Filling the buffer with crafted data to overwrite LR
- Construct JOP chain to defeat PXN
  - Mandatory on all Qualcomm arm64 devices
  - No RWX direct mapped pages (aka. Not Mediatek ;-))
- Not an easy one
  - Controlling X29 and X19 instead of conventional ones
  - Used a modified Ropper (@s4sh_s) to generate gadgets and search for a chain
  - https://github.com/idl3r/Ropper

# How to Exploit (cont.)

- Step 1: A pivot gadget (X19, X29 -> X0, X1)

```
bin_page_mkwrite:
            A1 1F 40 F9             LDR         X1, [X29,#0x38]
            E0 03 14 AA             MOV         X0, X20
            60 02 3F D6             BLR         X19
```

- Step 2: Conventional gadget sets for SP leak and addr_limit overwrite

```
shm_sync:
            05 08 40 F9             LDR         X5, [X0,#0x10]
            A0 14 40 F9             LDR         X0, [X5,#0x28]
            04 38 40 F9             LDR         X4, [X0,#0x70/0x78]
            A0 02 80 12             MOV         W0, #0xFFFFFFEA
            64 00 00 B4             CBZ         X4, loc_FFFFFFC0003DFB10
            E0 03 05 AA             MOV         X0, X5
            80 00 3F D6             BLR         X4

snd_pcm_common_ioctl1:
            03 08 40 F9             LDR         X3, [X0,#0x10]
            E0 03 1C AA             MOV         X0, X28
            60 00 3F D6             BLR         X3

__spi_async:
            20 08 00 F9             STR         X0, [X1,#0x10]
            22 34 00 B9             STR         W2, [X1,#0x34]
            A2 78 41 F9             LDR         X2, [X5,#0x2F0/0x380]
            40 00 3F D6             BLR         X2
```

```
shm_sync:
            05 08 40 F9             LDR         X5, [X0,#0x10]
            A0 14 40 F9             LDR         X0, [X5,#0x28]
            04 38 40 F9             LDR         X4, [X0,#0x70/0x78]
            A0 02 80 12             MOV         W0, #0xFFFFFFEA
            64 00 00 B4             CBZ         X4, loc_FFFFFFC0003DFB10
            E0 03 05 AA             MOV         X0, X5
            80 00 3F D6             BLR         X4

df_bcc_func:
            03 04 40 F9             LDR         X3, [X0,#8]
            00 18 40 F9             LDR         X0, [X0,#0x30]
            60 00 3F D6             BLR         X3

__spi_async:
            20 08 00 F9             STR         X0, [X1,#0x10]
            22 34 00 B9             STR         W2, [X1,#0x34]
            A2 78 41 F9             LDR         X2, [X5,#0x2F0/2F8/380]
            40 00 3F D6             BLR         X2
```

# Agenda

- Overview
  - Wi-Fi chipsets for Android
  - WEXT Attack Surface Analysis
  - Use device specific vulnerabilities to root them all
- Case Studies
  - Stack overflow vulnerability in Qualcomm WEXT
  - **Data section overflow vulnerability in MTK WEXT**
  - Use-After-Free vulnerability in Broadcom WEXT
- Google’s latest mitigation
- Conclusion

Tencent

KEEN security lab

# Case study 2 - CVE-2016-0820

- Data section overflow in MediaTek WEXT
- Discovered by KeenLab in Oct.2015 but not reported at that time
  - Obviously exploitable, **NO** hardcoded kernel symbol is needed
  - The exploit was finished in two days.
- Reported to Google by another researcher Mark Brand of Google P0 in Dec. 2015
- Affected all mediatek-based devices.

# Case study 2 - The overflow

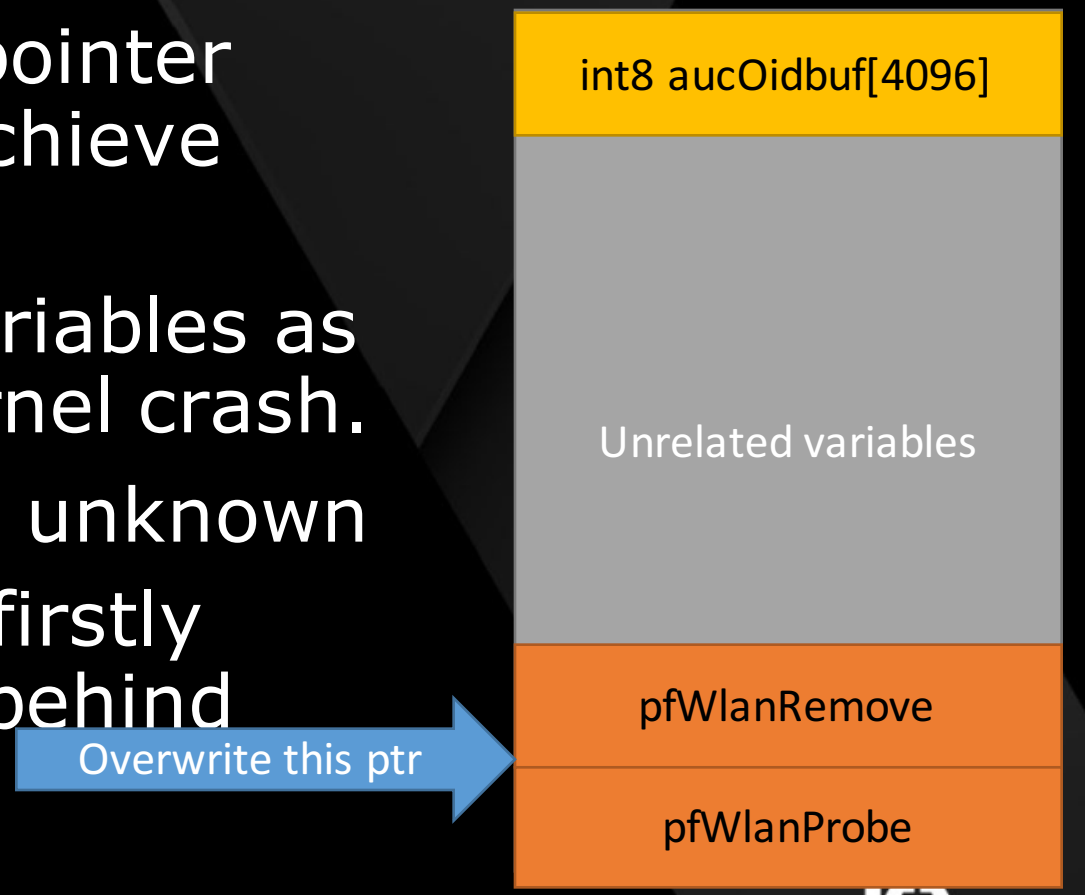
- No boundary protection of the copy length when *priv_get_struct* call *copy_from_user*()
- Destination: *aucOidbuf* has 4096 bytes in data section
- *prIwReqData->data.length* is provided by user, may be any value.

```
1437        case PRIV_CMD_SW_CTRL:
1438            pu4IntBuf = (PUINT_32)prIwReqData->data.pointer;
1439            prNdisReq = (P_NDIS_TRANSPORT_STRUCT) &aucOidBuf[0];
1440            //kalMemCopy(&prNdisReq->ndisOidContent[0],
    prIwReqData->data.pointer, 8);
1441            if (copy_from_user(&prNdisReq->ndisOidContent[0],
1442                               prIwReqData->data.pointer,
1443                               prIwReqData->data.length)) {
1444                status = -EFAULT;
1445                break;
1446            }
```

drivers/misc/mediatek/connectivity/combo/drv_wlan/mt6628/wlan/os/linux/gl_wext_priv.c

# Case study 2 – How to exploit

- Overwriting a global function pointer located behind *aucOidbuf* to achieve kernel code execution
- Corrupting unrelated global variables as little as possible to avoid a kernel crash.
- The offset of *pfWlanRemove* is unknown
- To meet these requirements , firstly leaking the value of variables behind *aucOidbuf* is necessary.



# Case study 2 – Leaking the value

- Another command PRIV_CMD_GET_DEBUG_CODE completed the task perfectly…
- No boundary check when call *copy_to_user* , data leaked.
- Now we get the value of variables behind *gucBufDbgCode* which is a variable just behind *aucOidbuf*

```
1097    case PRIV_CMD_GET_DEBUG_CODE:
1098        {
1099            wlanQueryDebugCode(prGlueInfo->prAdapter);
1100                kalMemSet(gucBufDbgCode, '.', sizeof(gucBufDbgCode));
1101                if (copy_to_user(prIwReqData->data.pointer, gucBufDbgCode,
    prIwReqData->data.length)) {
1102                    return -EFAULT;
1103                }
1104                else
1105                    return status;
1106        }
```

drivers/misc/mediatek/connectivity/combo/drv_wlan/mt6628/wlan/os/linux/gl_wext_priv.c

# Case study 2 – kernel code execution

- Copy shellcode to pages allocated in user space
- Get the direct mapped address of these pages in kernel (ret2dir), pages are **EXECUTABLE** in kernel space on MTK devices
- Overwrite *plWlanRemove* with kernel address of shellcode
- Call Java API ***wifi.setWifiEnabled****(false)* so that system process "mtk_wmtd" may call *plWlanRemove* to execute shellcode in kernel space
- Gain root and recover every modified global variables

# Agenda

- Overview
  - Wi-Fi chipsets for Android
  - WEXT Attack Surface Analysis
  - Use device specific vulnerabilities to root them all
- Case Studies
  - Stack overflow vulnerability in Qualcomm WEXT
  - Data section overflow vulnerability in MTK WEXT
  - **Use-After-Free vulnerability in Broadcom WEXT**
- Google's latest mitigation
- Conclusion

Tencent

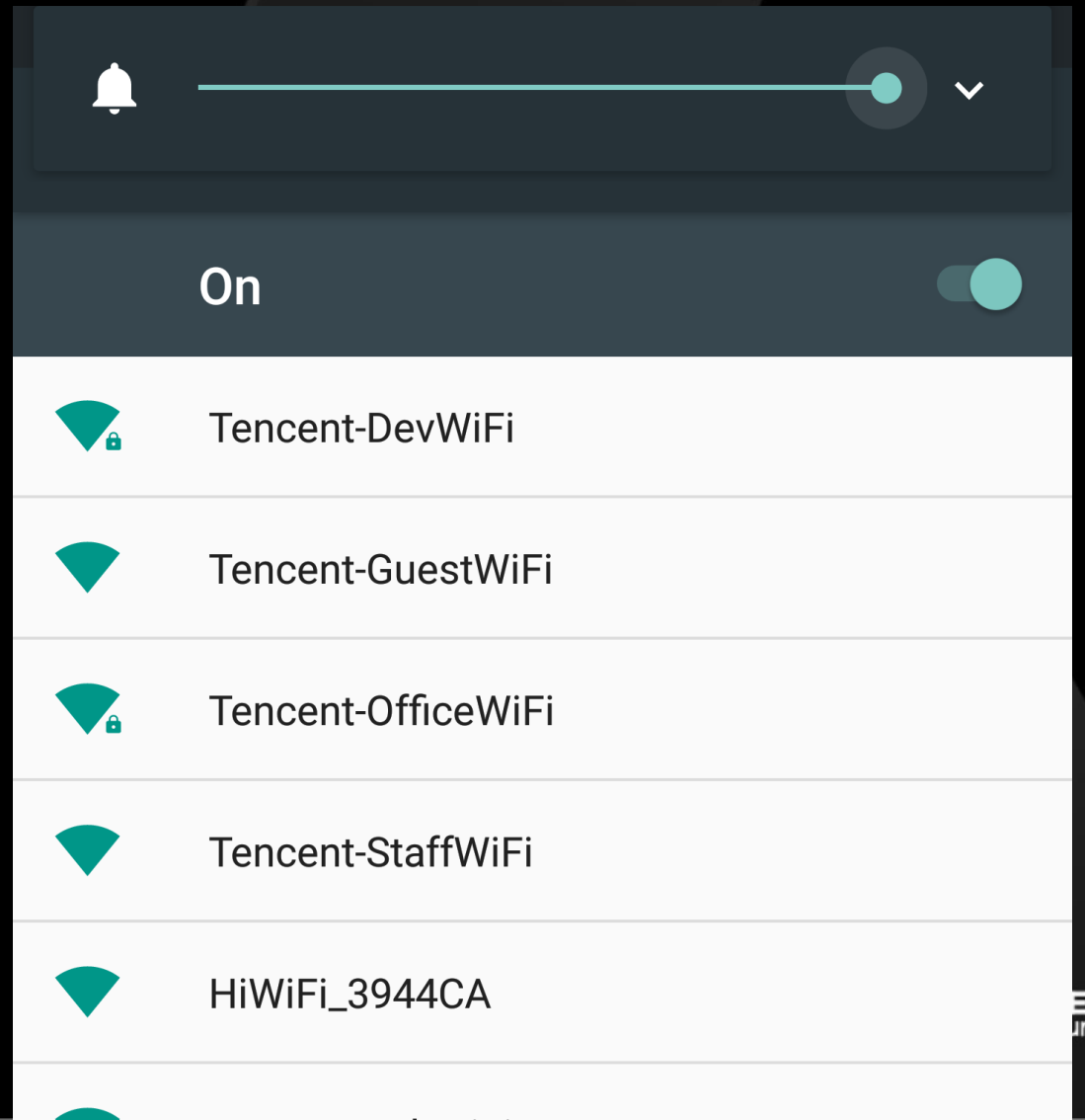
KEEN security lab

# Case study 3 – The Broadcom bugs

- Use-after-free due to race condition
  - Much complicated than previous two case
  - The window is small, need to refill the freed object in very short time
- Two separated issue
  - CVE-2016-2475: A lack of privilege check while processing WEXT ioctl cmd for Android.
  - Android-ID-24739315: Use-after-free when call *wl_android_wifi_off* concurrently
- Affected all premium-end Android phone like Samsung Galaxy series, Huawei Mate series, Google Nexus 6p,etc.

# Case study 3 – Discover the bug

- Discovered by running test code while pressing Wi-Fi button on and off repeatedly and crazily
- And then kernel crashed and the crash is reproducible…
- Analyzed the crash and found a UAF bug!



# Trigger a crash

- Thread-128 is the name of binder thread in system_server
- UAF due to race condition

```
<1>[  872.503389] Unable to handle kernel NULL pointer dereference at virtual address 0000004c
<1>[  872.503400] pgd = ffffffc0565ab000
<1>[  872.503408] [0000004c] *pgd=0000000000000000
<0>[  872.503429] Internal error: Oops: 96000005 [#1] PREEMPT SMP
<4>[  872.503444] CPU: 2 PID: 7137 Comm: Thread-128 Tainted: G        W    3.10.73-g42d0df9-dirty #111
<4>[  872.503456] task: ffffffc02c248ac0 ti: ffffffc056bf4000 task.ti: ffffffc056bf4000
<4>[  872.503469] PC is at sb_corereg+0x228/0x34c
<4>[  872.503481] LR is at sb_corereg+0x110/0x34c
<4>[  872.503489] pc : [<ffffffc0006632d4>] lr : [<ffffffc0006631bc>] pstate: 80000145
<4>[  872.503497] sp : ffffffc056bf7aa0
<4>[  872.503505] x29: ffffffc056bf7aa0 x28: ffffffc058ae1100
<4>[  872.503521] x27: 0000000000000000 x26: 0000000000000000
<4>[  872.503536] x25: 0000000000ab0300 x24: 00000000ff54fcff
<4>[  872.503553] x23: 0000000000000003 x22: ffffffc0ac815000
<4>[  872.503570] x21: 0000000000000000 x20: 000000000000004c
<4>[  872.503586] x19: 000000000000004c x18: 0000000000000005
<4>[  872.503602] x17: 0000000000000084 x16: 0000000000000001
<4>[  872.503618] x15: 0000000000000000 x14: 7475627322206 56c
<4>[  872.503633] x13: 6966203a22295d78 x12: 646965726f635b73
<4>[  872.503650] x11: 6765723e2d6f666e x10: 695f7365726f6328
<4>[  872.503666] x9 : 53474552444f4f47 x8 : 22205d3536333330
<4>[  872.503682] x7 : 352e32373820205b x6 : fffff8001b62815
<4>[  872.503698] x5 : fffff8000980000 x4 : 0000000000000000
<4>[  872.503713] x3 : 0000000000000000 x2 : ffffffc056bf4000
<4>[  872.503728] x1 : 000000000000
0001 x0 : 0000000000000001
<4>[  872.503756]
<0>[  872.503766] Proces  Thread-128 (pid: 7137, stack limit = 0xffffffc056bf4060)
<4>[  872.503775] Call trace:
<4>[  872.503786] [<ffffffc0006632d4>] sb_corereg+0x228/0x34c
<4>[  872.503797] [<ffffffc0006649d4>] si_corereg+0x14/0x68
<4>[  872.503810] [<ffffffc00069a430>] dhdpcie_bus_intr_disable+0x7c/0xb8
<4>[  872.503822] [<ffffffc00069d8fc>] dhd_bus_devreset+0x124/0x3c4
<4>[  872.503834] [<ffffffc0006531ec>] dhd_net_bus_devreset+0x8c/0xd4
<4>[  872.503846] [<ffffffc000668224>] wl_android_wifi_off+0x98/0xd0
<4>[  872.503859] [<ffffffc000656bc0>] dhd_stop+0x6c/0x17c
<4>[  872.503870] [<ffffffc000add678>] __dev_close_many+0x98/0xc0
<4>[  872.503882] [<ffffffc000add6c4>] __dev_close+0x24/0x40
<4>[  872.503894] [<ffffffc000ae0b90>] __dev_change_flags+0xb8/0x13c
<4>[  872.503906] [<ffffffc000ae0c90>] dev_change_flags+0x18/0x5c
<4>[  872.503920] [<ffffffc000b724c4>] devinet_ioctl+0x31c/0x690
<4>[  872.503931] [<ffffffc000b73af0>] inet_ioctl+0xc4/0xf4
<4>[  872.503944] [<ffffffc000ac8920>] sock_do_ioctl+0x2c/0x5c
<4>[  872.503956] [<ffffffc000ac8dc8>] sock_ioctl+0x208/0x228
<4>[  872.503969] [<ffffffc00030f728>] do_vfs_ioctl+0x48c/0x564
<4>[  872.503981] [<ffffffc00030f85c>] SyS_ioctl+0x5c/0x88
```

# First issue: Expose a surface for attacker

- CVE-2016-2475
- *wl_android_priv_cmd* is able to be called with insufficient privileges

Progressing SIOCDEVPRIVATE +1 without a check

Permission check, but too late

```
1  static int dhd_ioctl_entry(struct net_device *net,
   struct ifreq *ifr, int cmd)
2  {
3
4      ...snip...
5
6      if (cmd == SIOCDEVPRIVATE+1) {   //position 1
7          ret = wl_android_priv_cmd(net, ifr, cmd);
8          dhd_check_hang(net, &dhd->pub, ret);
9          DHD_OS_WAKE_UNLOCK(&dhd->pub);
10         return ret;
       }
12
13     if (cmd != SIOCDEVPRIVATE) {
14         DHD_PERIM_UNLOCK(&dhd->pub);
15         DHD_OS_WAKE_UNLOCK(&dhd->pub);
16         return -EOPNOTSUPP;
17     }
18
19     ...snip...
20
21     if (!capable(CAP_NET_ADMIN)) {     //position 2
22         bcmerror = BCME_EPERM;
           goto done;
24     }
25
26     ...snip...
27
28     bcmerror = dhd_ioctl_process(&dhd->pub, ifidx,
   &ioc, local_buf);
```

This issue was discovered by Keenlab in Nov. 2015,
firstly reported to Google by anonymous researcher in Jan. 2016.

# A large number of commands can be progressed here...

- Any local application can use CMD_START / CMD_STOP to enable or disable Wi-Fi devices directly.
- And here comes the second issue , the UAF bug

```
1  int wl_android_priv_cmd(struct net_device *net, struct ifreq
   *ifr, int cmd){
2      ...snip...
3      if (strnicmp(command, CMD_START, strlen(CMD_START)) == 0) {
4          bytes_written = wl_android_wifi_on(net);
5      }
6      else if (strnicmp(command, CMD_SETFWPATH,
   strlen(CMD_SETFWPATH)) == 0) {
7          bytes_written = wl_android_set_fwpath(net, command,
   priv_cmd.total_len);
8      }
9      if (!g_wifi_on) {
10         ret = 0;
11         goto exit;
12     }
13     if (strnicmp(command, CMD_STOP, strlen(CMD_STOP)) == 0) {
14         bytes_written = wl_android_wifi_off(net, FALSE);
15     }
16     else if (strnicmp(command, CMD_SCAN_ACTIVE,
   strlen(CMD_SCAN_ACTIVE)) == 0) {
17         ...snip...
18     }
19     else if (strnicmp(command, CMD_SCAN_PASSIVE,
   strlen(CMD_SCAN_PASSIVE)) == 0) {
20         ...snip...
21     }
22     else if (strnicmp(command, CMD_RSSI, strlen(CMD_RSSI)) == 0)
   {
23         ...snip...
24     }
25     ...snip...
26     return ret;
27 }
```

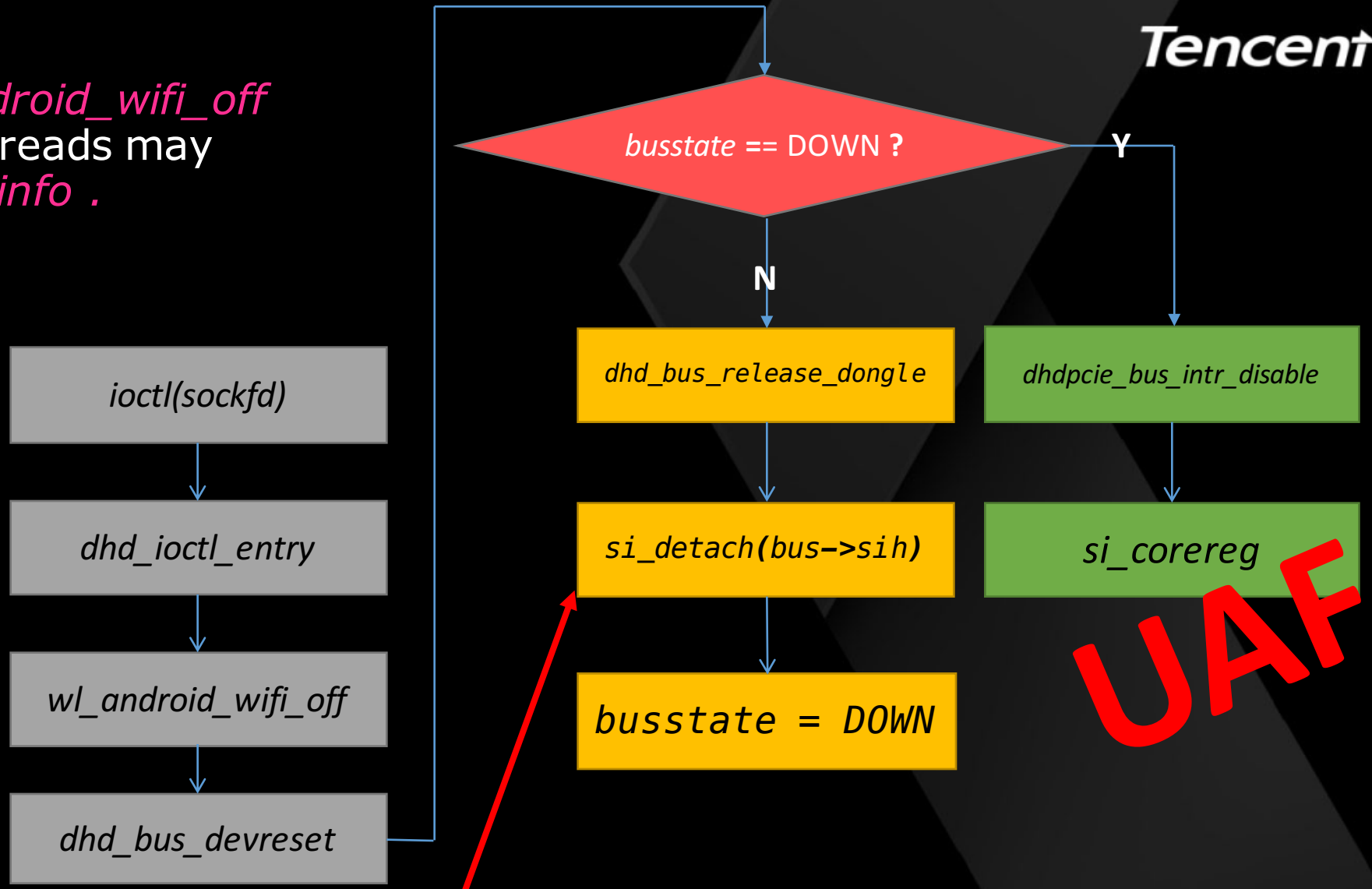
# Android-ID-24739315

- The patch is quite simple. When *dhd_bus_devreset* is called and the state of Wi-Fi bus is down, do not call *dhdpcie_bus_intr_disable* any more
- No CVE assigned, never appeared in Android Security Bulletin, but absolutely exploitable

```
net: wireless: bcmdhd: remove unnecessary PCIe memory access when BUS down.

In case PCIe BUS already down, we're not supposed to do access BAR0
area in any reason. One instance seen on test that made kernel panic.
removed disable irq calling which is useless in bus down case.

bug=24739315

Change-Id: I474e08c14c4dec0f4cc4cd207f29fef32e85ead7
Signed-off-by: Insun Song <isong@broadcom.com>

diff --git a/drivers/net/wireless/bcmdhd/dhd_pcie.c b/drivers/net/wireless/bcmdhd/dhd_pcie.c
index 21bbe54..1adffc3 100644
--- a/drivers/net/wireless/bcmdhd/dhd_pcie.c
+++ b/drivers/net/wireless/bcmdhd/dhd_pcie.c

@@ -2472,7 +2472,6 @@
                                bus->dhd->busstate = DHD_BUS_DOWN;
                        } else {
                                if (bus->intr) {
-                                       dhdpcie_bus_intr_disable(bus);
                                        dhdpcie_free_irq(bus);
                                }
```

This issue was discovered by Keenlab in Nov. 2015.
When we decide to report it in Feb. 2016, we noticed that a patch has already been released on public repository.

- If two threads call *wl_android_wifi_off* simultaneously, one of threads may reference freed *struct si_info* .




Branch for thread1

Branch for thread2

**bus->sih is freed by thread 1**

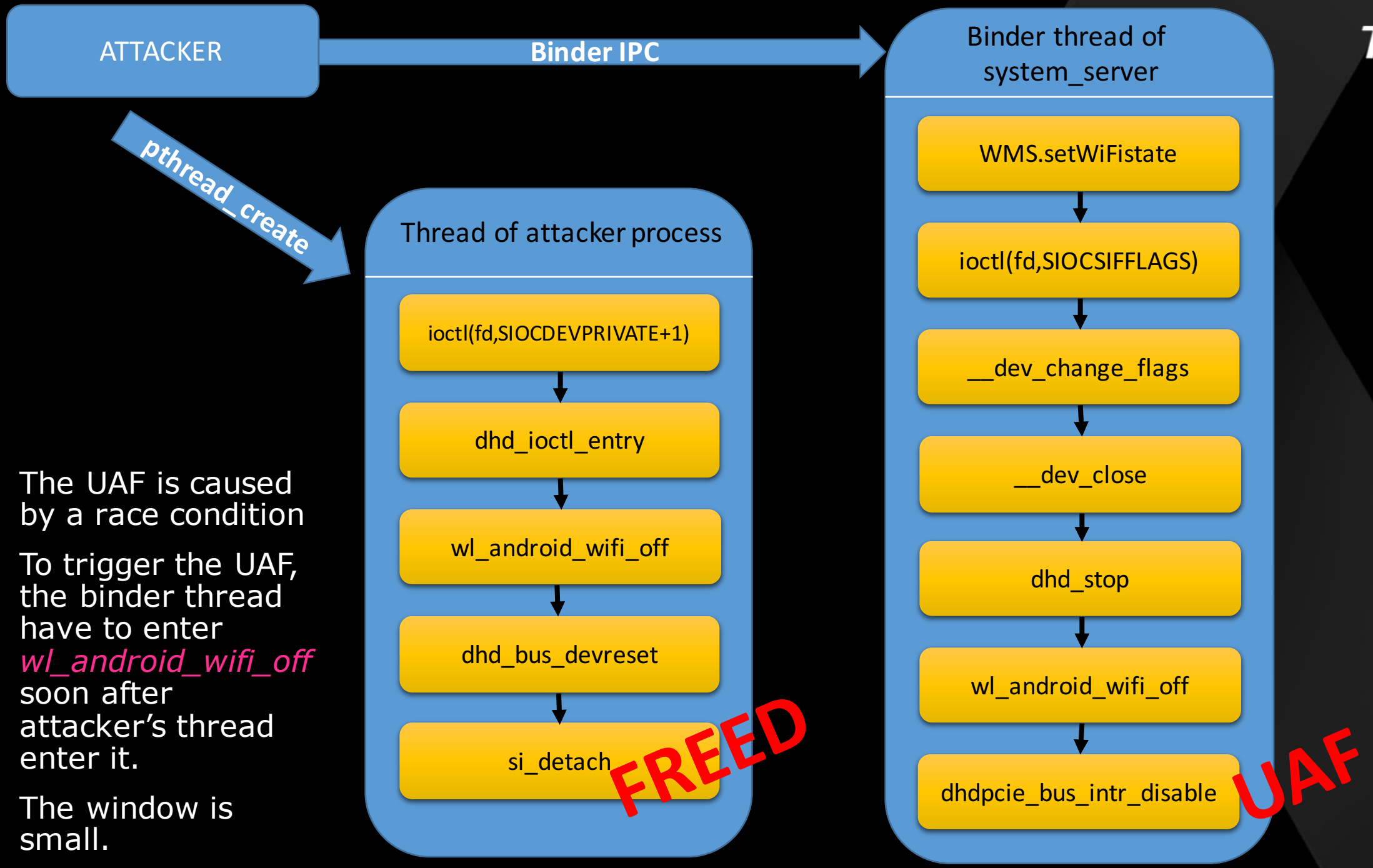
# How to trigger

- Unfortunately two threads can't invoke *wl_android_priv_cmd* concurrently because *dhd_ioctl_entry* is locked.
- Any other solution?

```
1  static int
2  dhd_ioctl_entry(struct net_device *net, struct ifreq *ifr, int
   cmd)
3  {
4      dhd_info_t *dhd = DHD_DEV_INFO(net);
5      dhd_ioctl_t ioc;
6      int bcmerror = 0;
7      int ifidx;
8      int ret;
9      void *local_buf = NULL;
10     u16 buflen = 0;
11
12     DHD_OS_WAKE_LOCK(&dhd->pub);
13     DHD_PERIM_LOCK(&dhd->pub);
14 ..snip..
15     if (cmd == SIOCDEVPRIVATE+1) {
16         ret = wl_android_priv_cmd(net, ifr, cmd);
17         dhd_check_hang(net, &dhd->pub, ret);
18         DHD_OS_WAKE_UNLOCK(&dhd->pub);
19         return ret;
20     }
21 ..snip..
22 done:
23     if (local_buf)
24         MFREE(dhd->pub.osh, local_buf, buflen+1);
25
26     DHD_PERIM_UNLOCK(&dhd->pub);
27     DHD_OS_WAKE_UNLOCK(&dhd->pub);
28 }
```

# How to trigger

- Is there a another way to invoke *wl_android_wifi_off* ?
  - Yes. *devnet_ioctl(sockfd,SIOCSIFFLAGS)* -> *__dev_change_flags* -> *__dev_close* -> *dhd_stop* -> *wl_android_wifi_off*
- Is *SIOCSIFFLAGS* able to be invoked by unprivileged process?
  - No… A CAP_NET_ADMIN is needed to do that.
- Is there any privileged process can do us a favor?
  - Yes. system_server is ready to serve!
  - Ask system_server via binder IPC to invoke *devnet_ioctl(sockfd,SIOCSIFFLAGS)* and trigger the UAF.




- The UAF is caused by a race condition
- To trigger the UAF, the binder thread have to enter *wl_android_wifi_off* soon after attacker's thread enter it.
- The window is small.

Tencent

KEEN security lab

```
1  void
2  dhdpcie_bus_intr_disable(dhd_bus_t *bus)
7      if (bus) {
8
9          if ((bus->sih->buscorerev == 2) || (bus->sih->buscorerev == 6) ||
10             (bus->sih->buscorerev == 4)) {
11             dhpcie_bus_mask_interrupt(bus);
12         }
13         else if (bus->sih) {
14             si_corereg(bus->sih, bus->sih->buscoreidx, PCIMailBoxMask,
15                 bus->def_intmask, 0);
16         }
17     }
19 }
```

buscorerev != 2,4,6

```
1  uint
2  si_corereg(si_t *sih, uint coreidx, uint regoff, uint mask, uint
   val)
3  {
4      if (CHIPTYPE(sih->socitype) == SOCI_SB)
5          return sb_corereg(sih, coreidx, regoff, mask, val);
6      else if ((CHIPTYPE(sih->socitype) == SOCI_AI) ||
   (CHIPTYPE(sih->socitype) == SOCI_NAI))
7          return ai_corereg(sih, coreidx, regoff, mask, val);
8      else if (CHIPTYPE(sih->socitype) == SOCI_UBUS)
9          return ub_corereg(sih, coreidx, regoff, mask, val);
10     else {
11         ASSERT(0);
12         return 0;
13     }
14 }
```

socitype == SOCI_SB

- *bus->sih is the freed si_info*

```
1  uint sb_corereg(si_t *sih, uint coreidx, uint regoff, uint mask,
   uint val)

2  {
3      ...snip...
4
5      if (BUSTYPE(sii->pub.bustype) == SI_BUS) {
6          fast = TRUE;
7          ...snip...
8      } else if (BUSTYPE(sii->pub.bustype) == PCI_BUS) {
9          fast = TRUE;
10         ...snip...
11     }
12
13     if (!fast) {
14         INTR_OFF(sii, intr_val);
```

bustype != SI_BUS,PCI_BUS

```
#define INTR_OFF(si, intr_val) \
    if ((si)->intrsoff_fn && (cores_info)->coreid[(si)->curidx]\ ==
(si)->dev_coreid) {    \

        intr_val = (*(si)->intrsoff_fn)((si)->intr_arg); }
```

- *si_info->intrsoff_fn* is the crafted function pointer I can control

# But how small the window is!

- Freed at *872.481513*
- Used at *872.592760*
- You have only 0.02s to re-fill the object…

```
1  <4>[  872.481513] si_detach free si ffffffc058ae1100;cores_info ffffffc0ac815000
2  <4>[  872.481526] dhdpcie_bus_release_dongle Exit
3  <4>[  872.481574] dhdpcie_stop_host_pcieclock Enter:
4  <6>[  872.496619] msm_pcie_disable: PCIe: Assert the reset of endpoint of RC1.
5  <4>[  872.501069] dhdpcie_stop_host_pcieclock Exit:
6  <4>[  872.501081] dhd_bus_devreset:  WLAN OFF Done
7  <4>[  872.501094] wifi_platform_set_power = 0
8  <6>[  872.501104] dhd_wlan_power Enter: power off
9  <4>[  872.501383] __dev_change_flags  dev: wlan0 flags 1042
10 <4>[  872.501598] dhd_deferred_work_handler: event to handle 24
11 <4>[  872.501615] dhd_set_mcast_list_handler: interface info not available/down
12 <4>[  872.501626] dhd_deferred_work_handler: event to handle 0
13 <4>[  872.501634] dhd_deferred_work_handler: No event to handle 0
14 <4>[  872.502660] dhd_stop: Enter ffffffc0b11d5000
15 <4>[  872.502672] wl_android_wifi_off in
16 <4>[  872.502684] dhd_prot_ioctl : bus is down. we have nothing to do
17 <4>[  872.502695] dhd_net_bus_devreset: wl down failed
18 <4>[  872.502704] dhd_bus_devreset: == Power OFF ==
19 <4>[  872.502715] dhdpcie_bus_intr_disable Enter
20 <4>[  872.502760] sb_corereg sii ffffffc058ae1100,cores_info ffffffc0ac815000
```

# Racing and object re-filling

- If racing failed
  - Nothing happened, try again
- If racing succeeded, but re-filling failed
  - Crash ):
- Need a way to spray kernel heap efficiently and quickly
- And also, in this case we'd better fully control the data and length of the re-filled objects

# Heap spraying by sendmmsg

- Create a TCP socket, two processes as server and client
- Send bytes over a TCP connection using sendmmsg
- Let msghdr->msg_control point to the content you want to spray in kernel.
- As server never respond the sendmsg request from client, the kernel buffer of msg_control will permanently stay in kmalloc heap.

```
1  struct msghdr {
2      void    *    msg_name;    /* Socket name            */
3      int         msg_namelen;    /* Length of name        */
4      struct iovec *    msg_iov;    /* Data blocks           */
5      __kernel_size_t    msg_iovlen;    /* Number of blocks        */
6      void     *    msg_control; /* Per protocol magic (eg BSD file descriptor passing) */
7      __kernel_size_t    msg_controllen;    /* Length of cmsg list */
8      unsigned int    msg_flags;
9  };
```

# Heap spraying by sendmmsg

- It has a 90% success rate to re-fill the freed object in 0.02s.
- The data and length of sprayed object can be fully controlled.
- Fortunately the freed *si_info* is allocated in kmalloc-256.
  - This approach will be not working if the object is located in kmalloc-512.
  - Sendmmsg will allocate other 512-sized object as an interference with spraying.

# Happy rooting

- Now you have kernel code execution
- Build JOP gadgets
- Manipulate credential of you task
- Disable SELinux
- Bypass some vendor specific mitigations

# Agenda

- Overview
  - Wi-Fi chipsets for Android
  - WEXT Attack Surface Analysis
  - Use device specific vulnerabilities to root them all
- Case Studies
  - Stack overflow vulnerability in Qualcomm WEXT
  - Data section overflow vulnerability in MTK WEXT
  - Use-After-Free vulnerability in Broadcom WEXT
- **Google's latest mitigation**
- Conclusion

# Google’s latest mitigation

- CVE-2016-0820 drove Google to reduce socket ioctl permissions further
- The same BUG ID with CVE-2016-0820

```
Reduce socket ioctl perms

Reduce the socket ioctl commands available to untrusted/isolated apps.
Neverallow accessing sensitive information or setting of network parameters.
Neverallow access to device private ioctls i.e. device specific
customizations as these are a common source of driver bugs.

Define common ioctl commands in ioctl_defines.

Bug: 26267358
Change-Id: Ic5c0af066e26d4cb2867568f53a3e65c5e3b5a5d
```

# Restrictions by SELinux policy

```
# only allow unprivileged socket ioctl commands
allow untrusted_app self:{ rawip_socket tcp_socket udp_socket } unpriv_sock_ioctls;

# Allow GMS core to access perfprofd output, which is stored
# in /data/misc/perfprofd/. GMS core will need to list all
```

- Only a limited set of ioctls are allowed to invoke by unprivileged process.

```
 # socket ioctls allowed to unprivileged apps
 define(`unpriv_sock_ioctls', `
 {
-# all socket ioctls except the Mac address SIOCGIFHWADDR 0x8927
-0x8900-0x8926 0x8928-0x89ff
-# all wireless extensions ioctls except get/set essid
-# IOCSIWESSID 0x8B1A SIOCGIWESSID 0x8B1B
-0x8B00-0x8B19 0x8B1C-0x8BFF
+# Socket ioctls for gathering information about the interface
+SIOCGIFNAME SIOCGIFCONF SIOCGIFFLAGS SIOCGIFADDR SIOCGIFBRDADDR
+SIOCGIFNETMASK SIOCGIFMTU SIOCGIFCOUNT SIOCGIFTXQLEN
+# Wireless extension ioctls. Primarily get functions.
+SIOCGIWNAME SIOCGIWFREQ SIOCGIWMODE SIOCGIWSENS SIOCGIWRANGE SIOCGIWPRIV
+SIOCGIWSTATS SIOCGIWSPY SIOCSIWTHRSPY SIOCGIWTHRSPY SIOCGIWRATE SIOCGIWRTS
+SIOCGIWFRAG SIOCGIWTXPOW SIOCGIWRETRY SIOCGIWPOWER
 # commonly used TTY ioctls
-0x5411 0x5451
+TIOCOUTQ FIOCLEX
 }')
```

- WEXT ioctls were removed from the set
- SELinux denied message

avc: denied { ioctl } for pid=8567 comm="poc" path="socket:[156925]" dev="sockfs" ino=156925 **ioctlcmd=89f1** scontext=u:r:shell:s0 tcontext=u:r:shell:s0 tclass=tcp_socket permissive=0

See also: [Ioctl command whitelisting with SELinux]

# Agenda

- Overview
  - Wi-Fi chipsets for Android
  - WEXT Attack Surface Analysis
  - Use device specific vulnerabilities to root them all
- Case Studies
  - Stack overflow vulnerability in Qualcomm WEXT
  - Data section overflow vulnerability in MTK WEXT
  - Use-After-Free vulnerability in Broadcom WEXT
- Google’s latest mitigation
- **Conclusion**

# Conclusion

- Once an awesome attack surface on Android kernel
- Vendor code is still buggy
- Google really did a good job on surface reduction in 2016
  - "Protecting Android with more Linux kernel defenses"
- Rooting Android is becoming more and more challenging
  - Mining another little known attack surface
  - Discovering another memory corruption vulnerability in generic syscall
  - Compromising a privileged process first (another hard work..)

Tencent

KEEN security lab

Tencent

KEEN
security
lab