How to Root 10^7 Phones with One Exploit

James Fang KEEN TEAM

About Me

- Root guy of Keen Team
  - With support from the whole team
- Responsible for many PHA (root) application
  - Pingpong Root
  - Mate 7 Root
  - KingRoot and SDK (with KR team)

Why Root?

- Fun
  - Xposed, Greenify, etc
  - New ideas from community (XDA, Gfan)
  - Full control over your device
- Secured
  - Root manger
  - Additional exploit mitigation

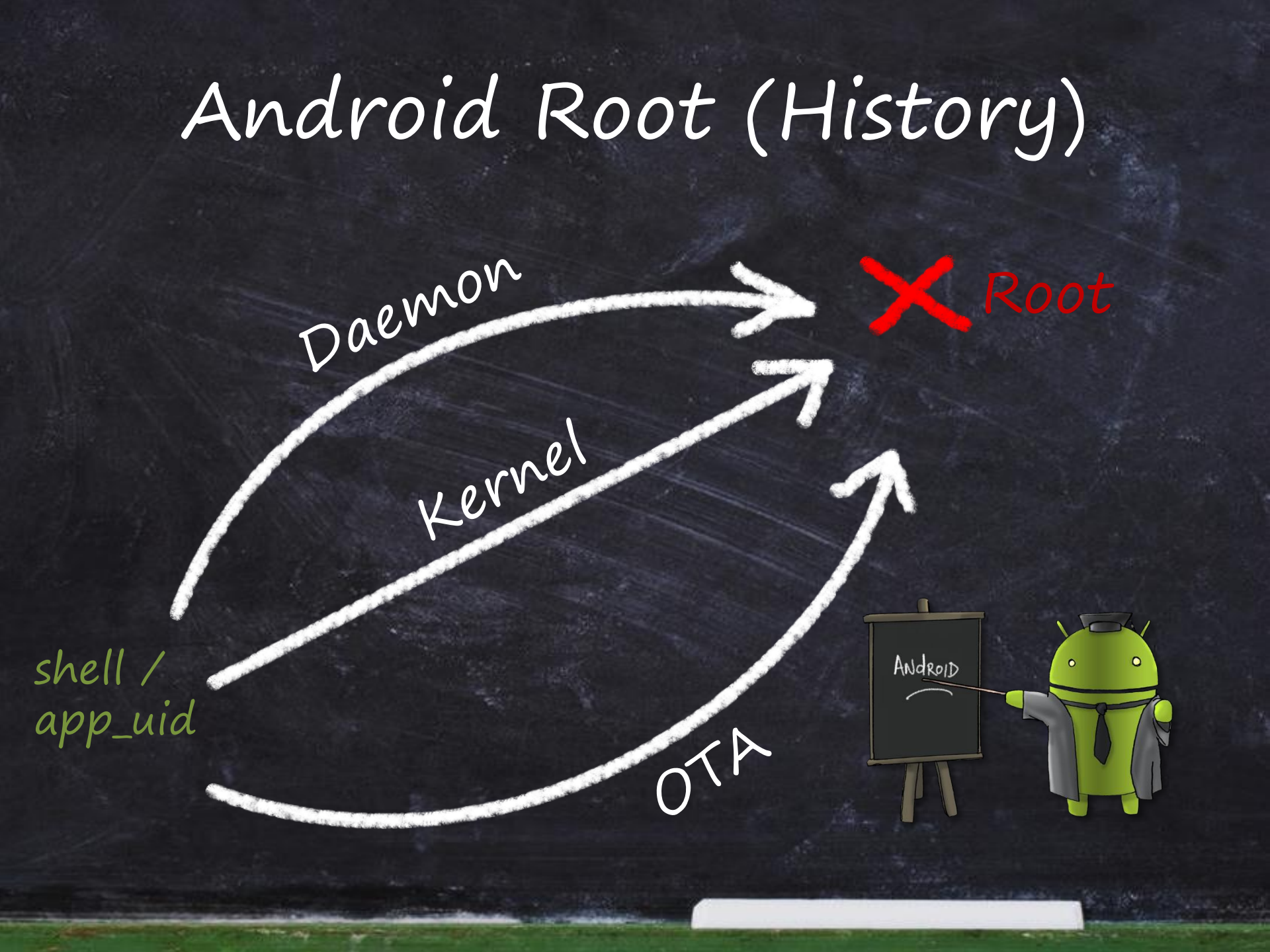
Android Root (History)

- 2008: T-Mobile G1 hidden root console
  - Run terminal emulator app
  - Launches telnetd as root

```
Trying 192.168.0.88...
Connected to 192.168.0.88.
Escape character is '^]'.
# id
uid=0(root) gid=0(root)
# cd proc
# cat version
Linux version 2.6.25-01843-gfea26b0 (android-build@apa27.mtv.corp.google.com)
(gcc version 4.2.1) #6 PREEMPT Mon Oct 6 14:13:36 PDT 2008
```

- Feature vs. bug

Android Root (History)



Android Root (History)

- OTA
  - Volez by Zinx
  - Improper parsing of signed update zip file
  - Master Key vulnerability "family"
    - Cydia impactor
    - system->root

Android Root (History)

- Daemon
  - setuid failure
  - Shared memroy
  - Memory corruption
    - GingerBreak
    - zergRush
  - File perms & symlinks ☺
    - Many were there in init.rc

Android Root (History)

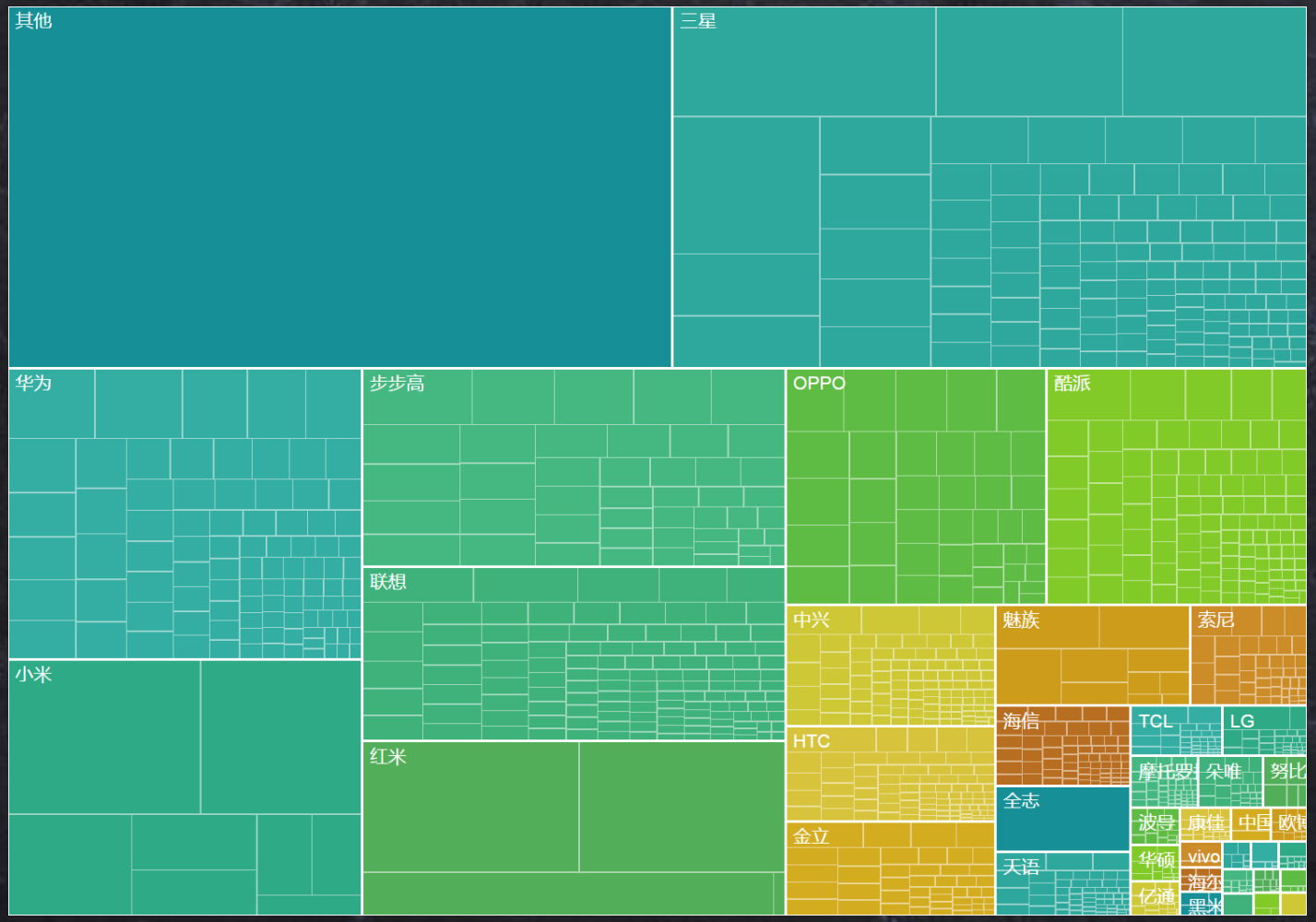
- Kernel
  - Wunderbar/asroot
  - FramaRoot by alphzain
    - /dev/exynosmem
    - /dev/exynosmem (again :D)
  - CVE-2013-6282
  - towelroot
  - …

How to ~~Cook~~ Root

Compounds

- Vulnerability & Exploit
  - Attack surface
  - In kernel/privileged process (to reach kernel)
- Mitigation Bypass
  - SELinux
  - Root mitigation (Samsung)
  - (NAND)? write protection

Challenges


其他
三星
华为
步步高
OPPO
酷派
联想
中兴
魅族
索尼
小米
海信
TCL
LG
HTC
摩托罗拉
朵唯
努比
红米
全志
波导
康佳
中国
欧
金立
华硕
vivo
天语
海尔
亿通
黑米

Challenges


三星Galaxy Note II
三星Galaxy S IV
三星Galaxy S III
三星Galaxy Note III
三星Galaxy Grand 2
三星Galaxy S V
三星GT-S7562
三星GT-S7568
三星Galaxy Ace
三星Galaxy Win
三星Galaxy S
三星Galaxy Note
三星Galaxy Mega 5.8
其他
三星Galaxy S II
三星Galaxy G
三星GT-i8262
三星i8268
三星i739
三星SM-G381
三星A5000
三星GT-S7
三星SCH-I8
三星GT-i
三星SM-G
三星n910
三星SCH
三星i879
三星SM-
三星R830

Challenges

- Manufacturers and models
  - Kernel behaviors (syscall?)
  - Root mitigation
- Same model, different ROM
  - Kernel symbol locations
  - Structures, offsets
- Device database?
  - Not the best solution

A Case Study (Early 2014)

- CVE-2013-6282
  - put_user/get_user
  - Lack of boundary check
- Read & write
  - ptrace->put_user->write anywhere
  - setsockopt->get_user->read anywhere (by getsockopt)
- Targeting all ARMv6+

A Case Study (Early 2014)

- "Perfect vulnerability"
- Thinking of these problems:
  - Achieve root with read & write
  - Achieve root with write only
  - Without device database
  - SELinux (Nexus and Samsung)
- "Write-only" bugs are quite common

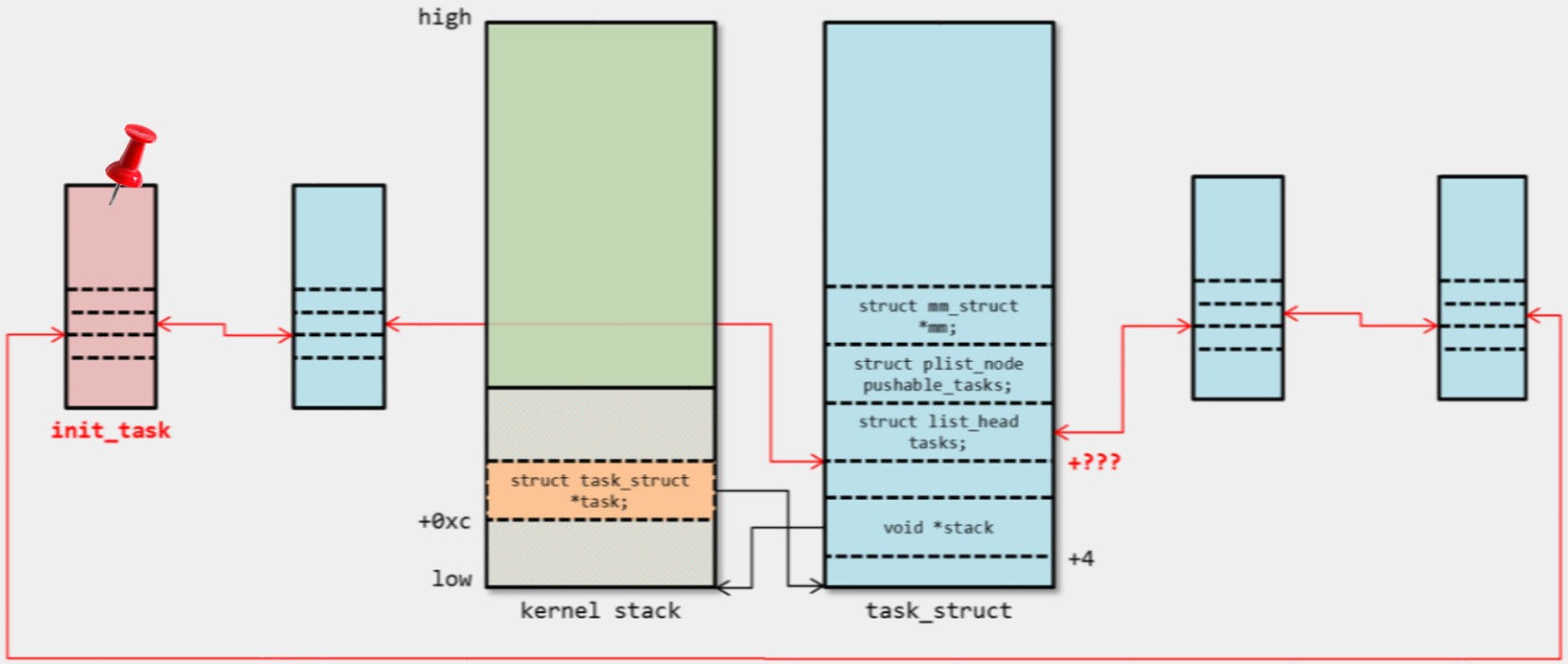
DKOM

- Direct Kernel Object Manipulation
- Straight forward when we have R/W
- Our goal
  - Get root uid/gid
  - Get full capabilities
  - Get u:r:init:s0
  - Patch SELinux policy
  - Patch/bypass root mitigations
- Where are the "objects"?

DKOM

- Task information
  - struct thread_info => on kernel stack
  - struct task_struct => in heap
  - thread_info.task -> task_struct
- Leak SP (towelroot)
- Search and manipulate
  - Useful when code execution is inconvenient

DKOM


high
low
init_task
struct task_struct
*task;
+0xc
kernel stack
struct mm_struct
*mm;
struct plist_node
pushable_tasks;
struct list_head
tasks;
+???
void *stack
+4
task_struct

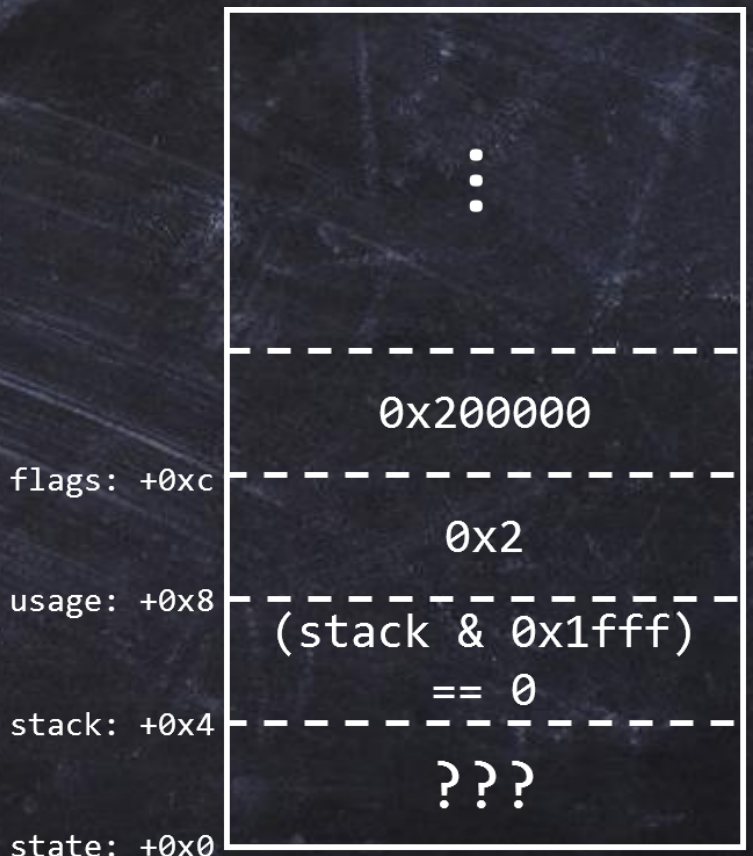
DKOM

- init_task leads to the doubly-linked list
- Usually a symbol in kallsyms
  - We don't have root yet
- Any pattern?
- In /arch/arm/kernel/init_task.c
  - struct task_struct init_task = INIT_TASK(init_task);

(take ARM32 as an example)

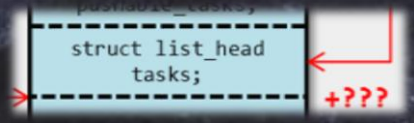
DKOM

- Always starts with this pattern:
  - .state => unknown
  - .stack => 2-page (8K) aligned address
  - .usage => 0x2
  - .flags => 0x200000
- Search range?
  - /proc/iomem



DKOM

- Next: offset of tasks
- Structure definition:




```
struct task_struct {
        volatile long state;
        void *stack;
...
        struct list_head tasks;
#ifdef CONFIG_SMP
        struct plist_node pushable_tasks;
#endif
```

- For all modern multi-core phones, CONFIG_SMP is always ‘y’

DKOM

- Check init_task.h, list.h and plist.h in /include/linux
- tasks + pushable_tasks looks like:
  - Two kernel pointers
  - 0x0000008c
  - Two kernel pointers
- Good anchor once we located init_task

DKOM

- How to identify my process?
  - Random string => char comm[TASK_COMM_LEN]
  - Anchor to identify task
  - It will also find task->cred for you ☺

```
const struct cred __rcu *cred;
struct cred *replacement_session_keyring;
char comm[TASK_COMM_LEN];
```

DKOM

- So now we have:
  - init_task
  - Traverse task list for any task
  - Identify task by comm
  - Offset of task->cred
- struct cred is simple and easy?
  - For uid/gid
  - How about u:r:init:s0?

DKOM

- sid <=> SELinux context
- 1 <=> u:r:kernel:s0
- ?? <=> u:r:init:s0
- Two approaches
  - untrusted/shell => kernel => init
    - Policy may block it
  - Set sid directly
- Look for "init" task ☺

```
                    -*> MEMORY DUMP <*-
-+------------------------------------------------+-
 |  0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F|
-+------------------------------------------------+-
 | 01 00 00 00 30 00 00 00  00 00 00 00 00 00 00 00 |
 | 00 00 00 00 00 00 00 00                          |
-+------------------------------------------------+-
```

-wx-wx-wx

- But what if only write anywhere is possible?
- Real example:
  - Broadcom /dev/uio1
  - ioctl: Write a string to given address
  - How to exploit?
- Overwrite syscall table and run shell code?

-wx-wx-wx

- Interesting entry in process map:

```
shell@test:/ $ cat /proc/self/maps
40000000-40020000 r--s 00000000 00:0b 4103      /dev/__properties__
40031000-40033000 rw-p 00000000 00:00 0
4006d000-40092000 r-xp 00000000 b3:1c 353       /system/bin/mksh
40092000-40093000 r--p 00000000 00:00 0
.
.
.
401c1000-401c4000 rw-p 0004a000 b3:1c 1704      /system/lib/libc.so
401c4000-401d2000 rw-p 00000000 00:00 0
4030e000-40319000 rw-p 00000000 00:00 0         [heap]
bef49000-bef6a000 rw-p 00000000 00:00 0         [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0         [vectors]
```

- What's inside?

-wx-wx-wx

- In /arch/arm/kernel/entry-armv.S

```
.LCvswi:
            .word       vector_swi
...
__vectors_start:
 ARM(       swi         SYS_ERROR0   )
 THUMB(     svc         #0                          )
 THUMB(     nop                                     )
            W(b)        vector_und + stubs_offset
            W(ldr)      pc, .LCvswi + stubs_offset
            W(b)        vector_pabt + stubs_offset
...
            W(b)        vector_fiq + stubs_offset

            .globl      __vectors_end
__vectors_end:
```

- At 0xffff0008, location of **vector_swi** is leaked by an LDR instruction.
- loc_vector_swi = *(0xffff0008 + 8 + ((*0xffff0008) & 0xfff))

-wx-wx-wx

- Symbol sys_call_table is always 3 symbols aways from vector_swi

```
vector_swi
__sys_trace
__sys_trace_return
__cr_alignment
=>sys_call_table
```

- All defined in /arch/arm/kernel/entry-common.S
- loc_sys_call_table = loc_vector_swi + len(4 following stubs)

-wx-wx-wx

- Length of vector_swi depends on:
  - CONFIG_OABI_COMPAT - Allow old ABI binaries to run with this kernel (EXPERIMENTAL)
  - CONFIG_SECCOMP - Enable seccomp to safely compute untrusted bytecode
- In most cases both are set as N.
- If set:
  - CONFIG_SECCOMP => 6 instructions, 24 bytes
  - CONFIG_OABI_COMPAT => More complex, 28 bytes

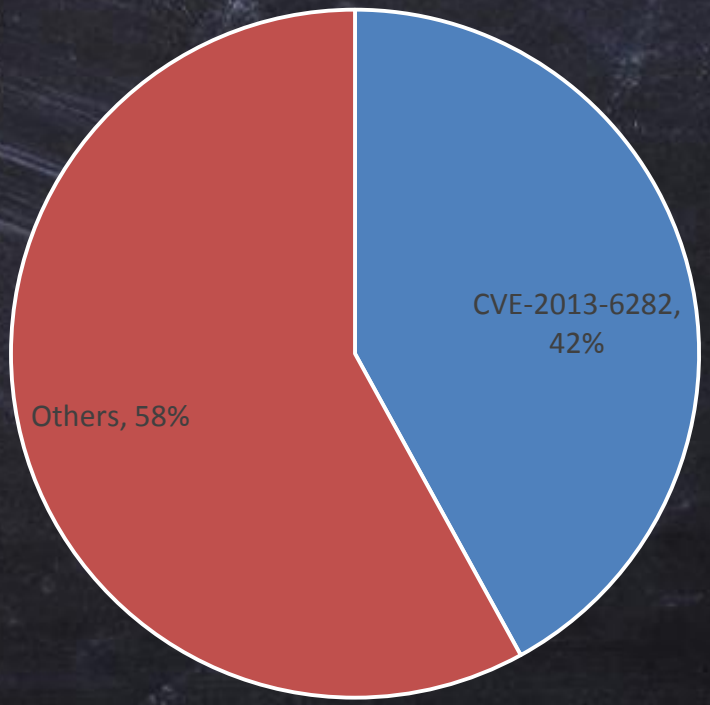
-WX-WX-WX

- CONFIG_SECCOMP: **PRCTL(PR_GET_SECCOMP, ...)**
- CONFIG_OABI_COMPAT: **sys_oabi_semop**
- Algorithm is simple
  - Set BASE_OFFSET
  - If (CONFIG_SECCOMP) BASE_OFFSET += 24
  - If (CONFIG_OABI_COMPAT) BASE_OFFSET += 28
  - Align
  - loc_sys_call_table = loc_vector_swi + BASE_OFFSET
- Patch unused syscall entry and run shell code
- Assuming no PXN or RO kernel text
  (True for most devices before 2014)

Conclusion

- A good vulnerability
- Hardcode-free exploitation
- Fine tune

=> Decent coverage

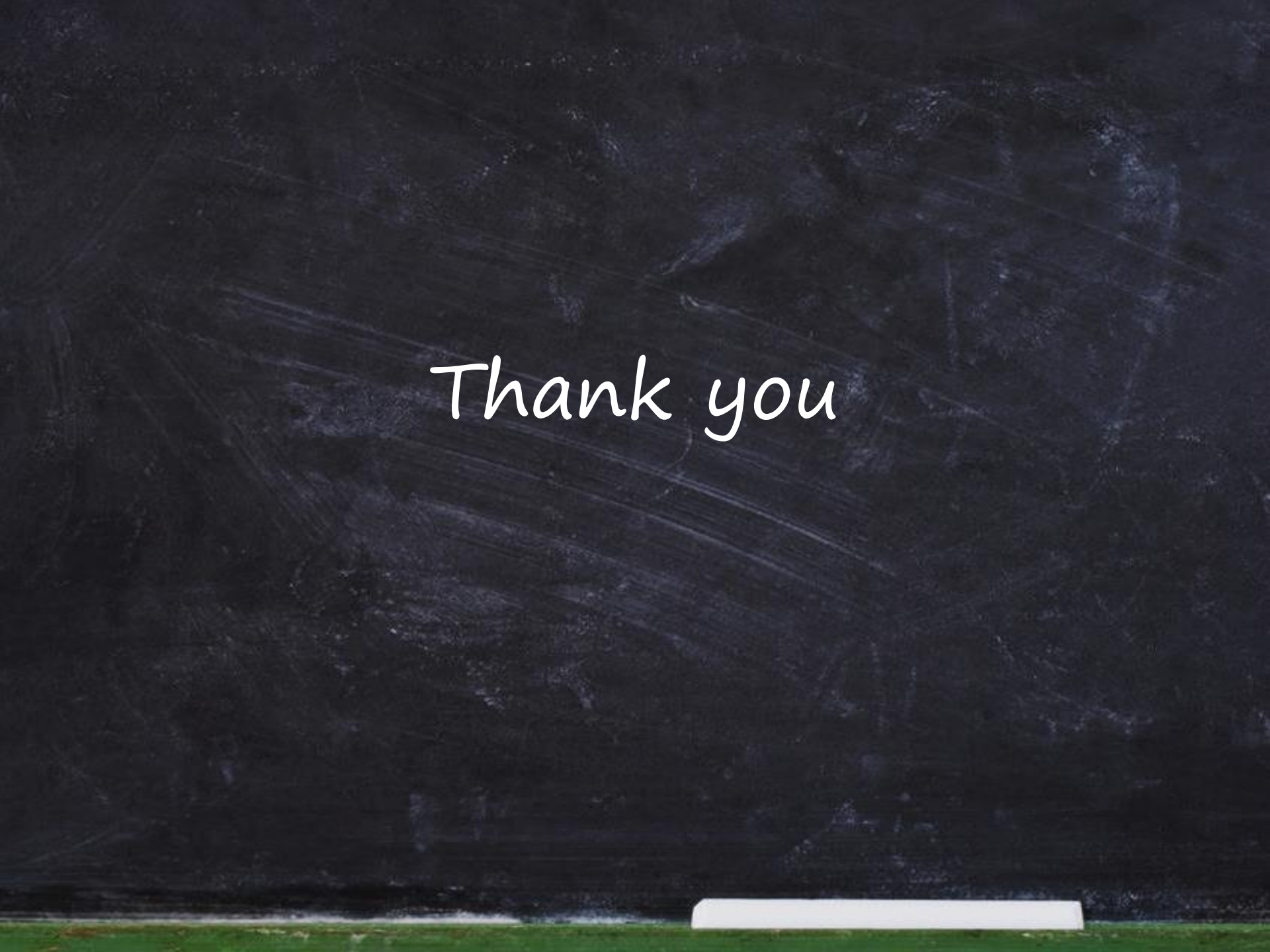


What's next?

- Root mitigation bypass
  - Samsung (STRICT_SEC)
  - Huawei fblock
  - Sony RIC
- CVE-2015-3636
  - How to root 10^7 phones again?
  - BH 2015

  Ah! Universal Android Rooting is Back

  Present by Wen Xu


Thank you