Tencent

Defeating Samsung KNOX with zero privilege

Di Shen a.k.a. Retme (@returnsme)
Keen Lab of Tencent

KEEN security lab

black hat USA 2017

whoami

- Di Shen a.k.a. Retme (@returnsme)
- Member of Keen Lab
- Android Kernel vulnerability hunting and exploitation since 2014
- Aim: to make out universal rooting exploit for Android
- Trophy:
  - CVE-2016-6787 & CVE-2017-0403 (kernel/events/core.c)
    - Credit to discoveries and exploits
  - CVE-2015-1805 (fs/pipe.c)
    - First working exploit
  - CVE-2015-4421,4422
    - Kernel LPE and TrustZone code execution for Huawei Mate 7
  - Exploiting Wireless Extension for all common Wi-Fi chipsets (BHEU 16')
  - And more To Be Announced in the future

Agenda

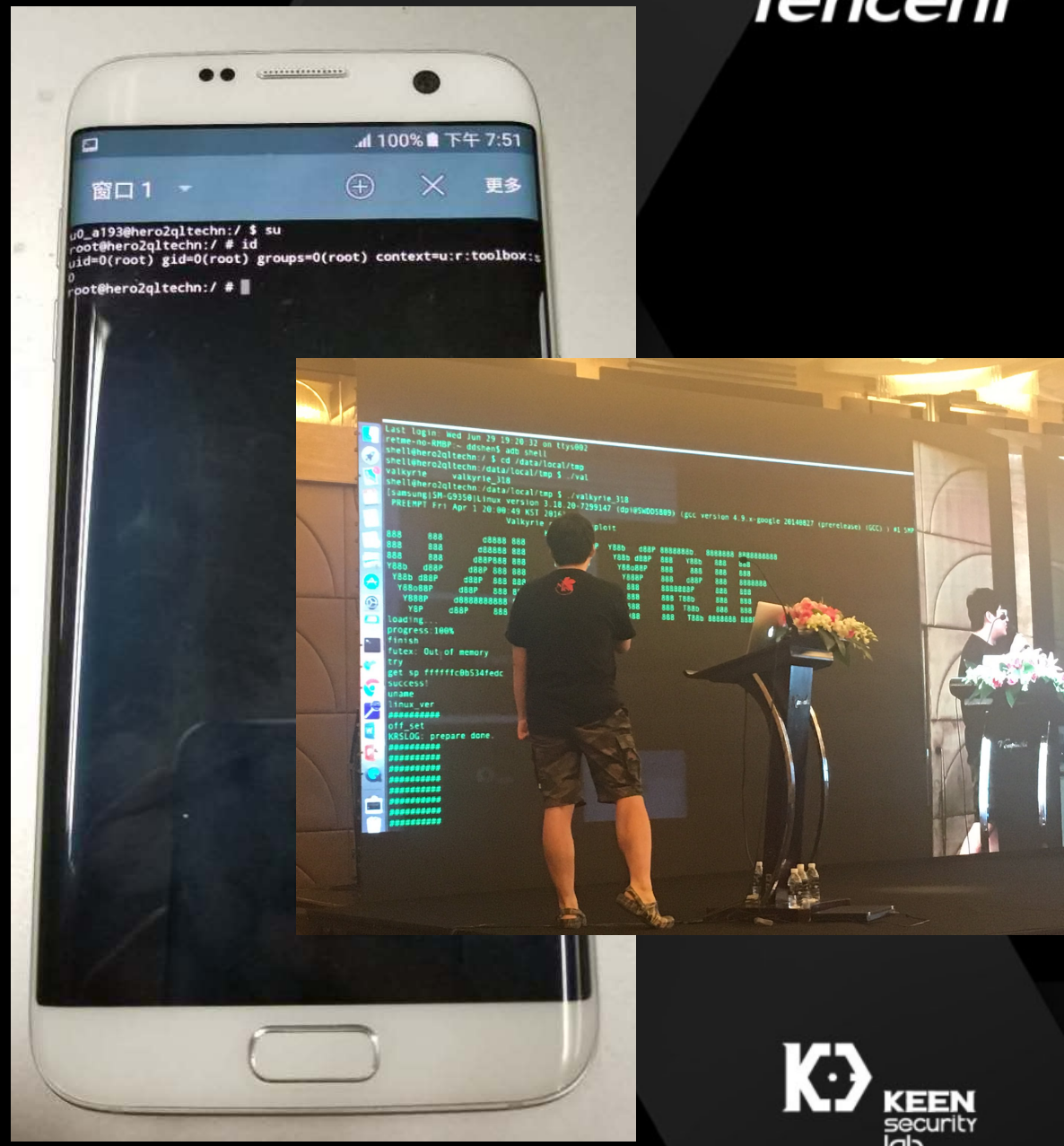
- Overview of KNOX 2.6
  - KASLR (Samsung’s implementation)
  - Real-time kernel protection (RKP)
  - Data Flow Integrity (DFI)
  - SELinux enhancement
- Bypassing techniques
  - KASLR bypassing
  - DFI bypassing
  - SELinux bypassing
  - Gain root

Tencent

KEEN security lab

Target device

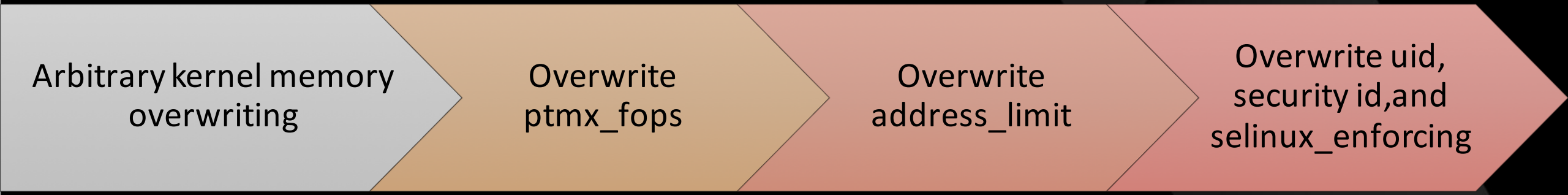
- Samsung Galaxy S7 edge
  - SM-G9350 (Hong Kong ver.)
  - Qualcomm-based
  - KNOX 2.6
- Exploit chain was finished in June 2016
- Demonstrated in July 1st 2016 at Shanghai



Tencent

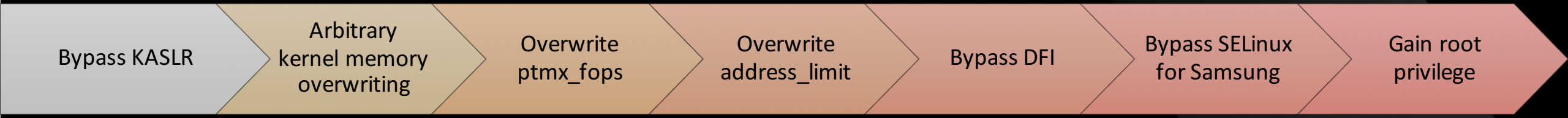
KEEN security lab

Common LPE flow on Android



LPE flow on Galaxy S7 edge

Tencent



KEEN security lab

KASLR for Linux 3.18 - Initialization

Tencent

- CONFIG_RELOCATABLE_KERNEL by Samsung
- The Random size is passed to kernel by loader
- X1,X2 are set upon kernel start up
  - X1: phycal offset X2: vitual text offset
  - Store to __boot_kernel_offset
  - __boot_kernel_offset[0] : physical address of kernel
  - __boot_kernel_offset[1] : the actual load address
  - __boot_kernel_offset[2] : TEXT_OFFSET 0x8000

```
ENTRY(stext)
#ifdef CONFIG_RKP_CFP_ROPP
    /* Must intialize RRK to zero before any RET/BL
    mov RRK, #0
#endif
#ifdef CONFIG_RKP_CFP_JOPP
    /* We need RRS to be loaded before we take our
    load_function_entry_magic_number_before_reloc h
#endif
#ifdef CONFIG_RELOCATABLE_KERNEL
    mov x22, x1              // x1=PHYS_OFFSET
    mov x19, x2              // x2=real TEXT_OFFSET
    adr x21, __boot_kernel_offset
    stp x1, x2, [x21]
#endif
```

KEEN security lab

KASLR for Linux 3.18 - relocating

Tencent

- __relocate_kernel() handles kernel relocating
  - Similar to a aarch64 linker in user space

```
#ifdef CONFIG_RELOCATABLE_KERNEL

#define R_AARCH64_RELATIVE  0x403
#define R_AARCH64_ABS64     0x101

__relocate_kernel:
    sub x23, x19, #TEXT_OFFSET
    adrp    x8, __dynsym_start
    add x8, x8, :lo12:__dynsym_start    //x8: start of symbol table
    adrp    x9, __reloc_start
    add     x9, x9, :lo12:__reloc_start //x9: start of relocation table
    adrp    x10, __reloc_end
    add x10, x10, :lo12:__reloc_end     //x10: end of relocation table
```

KEEN security lab

KASLR for Linux 3.18 - .rela section

Tencent

- __relocate_kernel handles kernel relocating
  - Similar to a aarch64 linker in user space
  - Relocation section '.rela' at offset 0x1446600 contains 233903 entries:

```
 Offset          Info           Type           Sym. Value    Sym. Name + Addend
ffffffc000081698  000000000403 R_AARCH64_RELATIV                 -3ffff7e968
ffffffc0000816a0  000000000403 R_AARCH64_RELATIV                 -3ffe5d1e20
ffffffc000081798  000000000403 R_AARCH64_RELATIV                 -3ffff7e868
...
ffffffc00008e000  000000000403 R_AARCH64_RELATIV                 -3ffff546b8     # Begin of sys_call_table
...
ffffffc000f6f800  000600000101 R_AARCH64_ABS64    ffffffc000080000 _text + 0      # Begin of kallsyms_addresses
...
ffffffc0013b1468  000000000403 R_AARCH64_RELATIV                 -3ffec1afdd
```

KEEN security lab

Bypassing KASLR

- Readable TIMA logs

```
shell@hero2qltechn:/proc $ ls -l | grep tz
shell@hero2qltechn:/proc $ ls -l | grep tima
-rw-r--r-- root     root              0 2016-05-14 16:52 tima_debug_log
-rw-r--r-- root     root              0 2016-05-14 16:52 tima_debug_rkp_log
-rw-r--r-- root     root              0 2016-05-12 19:44 tima_secure_rkp_log
shell@hero2qltechn:/proc $
```

Kernel information leaking

- Kernel pointer leaked in /proc/tima_secure_rkp_log
- At 0x13B80 -> *init_user_ns*
- Real:0xFFFFFFC001B0EFB8 Static:0xFFFFFFC01A3AFB8
- KASLR offset = 0xD4000

```
ROM:FFFFFFC01A3AFB8 init_user_ns    DCB     1
                                    DCB     0
ROM:FFFFFFC01A3AFBA                 DCB     0
ROM:FFFFFFC01A3AFBB                 DCB     0
ROM:FFFFFFC01A3AFBC                 DCB     0
ROM:FFFFFFC01A3AFBD                 DCB     0
ROM:FFFFFFC01A3AFBE                 DCB     0
ROM:FFFFFFC01A3AFBF                 DCB     0
ROM:FFFFFFC01A3AFC0                 DCB     0
ROM:FFFFFFC01A3AFC1                 DCB     0
ROM:FFFFFFC01A3AFC2                 DCB     0
```

```
1:3B00h: 0000000000000001 00
1:3B10h: 0000000000000000 00
1:3B20h: 0000000000000000 00
1:3B30h: 0000003FFFFFFFFF 00
1:3B40h: 0000003FFFFFFFFF 00
1:3B50h: 0000000000000000 0000000000000000  ................
1:3B60h: 0000000000000000 0000000000000000  ................
1:3B70h: FFFFFFC01C614400 FFFFFFC001B0EF60  ÿÿÿÿÀ.aD.ÿÿÿÿÀ.°ï`
1:3B80h: FFFFFFC001B0EFB8 FFFFFFC001B107E8  ÿÿÿÿÀ.°ï¸ÿÿÿÿÀ.±.è
1:3B90h: 0000000000000000 0000000000000000  ................
         FFFFFFC0E8AE0100 FFFFFFC0F783A400  ÿÿÿÿÀø®..ÿÿÿÿÀ÷ƒ¤.
         0000000000000000 0000000000000000  ÿÿÿÿÀ.Q ........
         FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF  ÿÿÿÿÀ.a<.ÿÿÿÿÿÿÿÿ
         FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

```
root@hero2qltechn:/ # cat /proc/kallsyms
ffffffc00148cab8 R __ksymtab_init_user_ns
ffffffc001496883 r __kstrtab_init_user_ns
ffffffc001b0efb8 D init_user_ns
```

Achieve arbitrary kernel mem overwriting

- By exploiting CVE-2016-6787
- Use-after-free due to race condition in perf subsystem
  - Moving group in sys_perf_event_open() is not locked by mutex correctly
- Spray struct perf_event_context{}
  - Control code flow by refill ctx->pmu->pmu_disable(X0)
- Another long story ☺

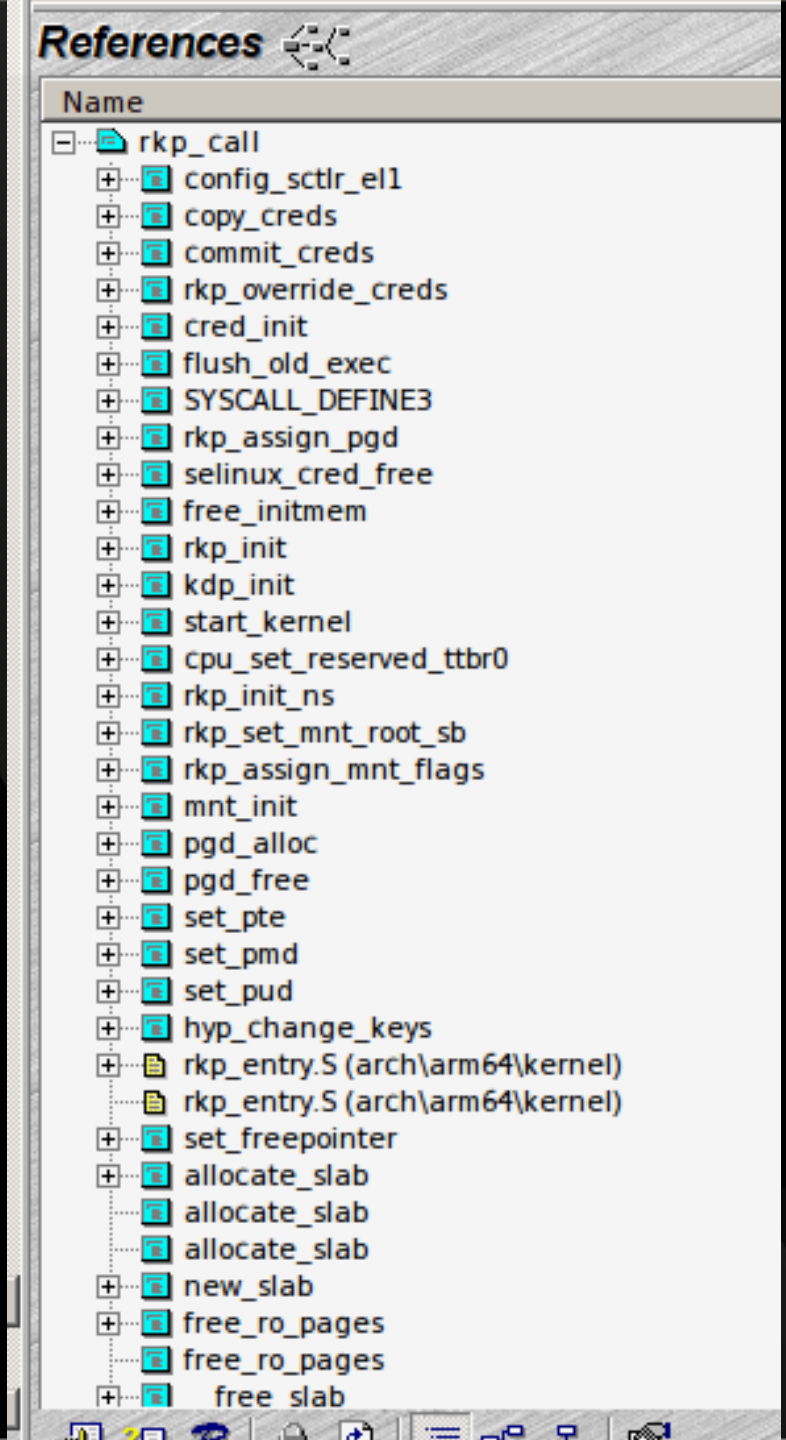
```
+       /*
+        * Take the group_leader's group_leader_mutex before observing
+        * anything in the group leader that leads to changes in ctx,
+        * many of which may be changing on another thread.
+        * In particular, we want to take this lock before deciding
+        * whether we need to move_group.
+        */
+       if (group_leader)
+               mutex_lock(&group_leader->group_leader_mutex);
+
        if (pid != -1 && !(flags & PERF_FLAG_PID_CGROUP)) {
                task = find_lively_task_by_vpid(pid);
                if (IS_ERR(task)) {
@@ -7686,6 +7697,8 @@
                put_ctx(gctx);
        }
        mutex_unlock(&ctx->mutex);
+       if (group_leader)
+               mutex_unlock(&group_leader->group_leader_mutex);
```

Real-time Kernel Protection

- Implemented in TrustZone or hypervisor
  - Depends on device model, for S7 edge (SM-G9350), it's TrustZone
- CONFIG_TIMA_RKP , CONFIG_RKP_KDP
- Targeted features via samsungknox.com:
  - "completely prevents running unauthorized privileged code"
  - "prevents kernel data from being directly accessed by user processes"
  - "monitors some critical kernel data structures to verify that they are not exploited by attacks"

rkp_call()

- RKP call entry
  - Called by many critical kernel functions
    - SLAB allocation and de-allocation
    - Page Table operations
    - Copy/Override/Commit creds



Kernel code protection

- Not exclusive features for KNOX 2.6
  - "config KERNEL_TEXT_RDONLY"
  - Data section not executable
  - Privileged eXecute Never (PXN)
    - Kill ret2user and other ancient tricks
- New in KNOX 2.8?
  - Control flow protection

PXNTable, bit[59]

PXN limit for subsequent levels of lookup, see *Hierarchical control of instruction fetching* page D5-1786.

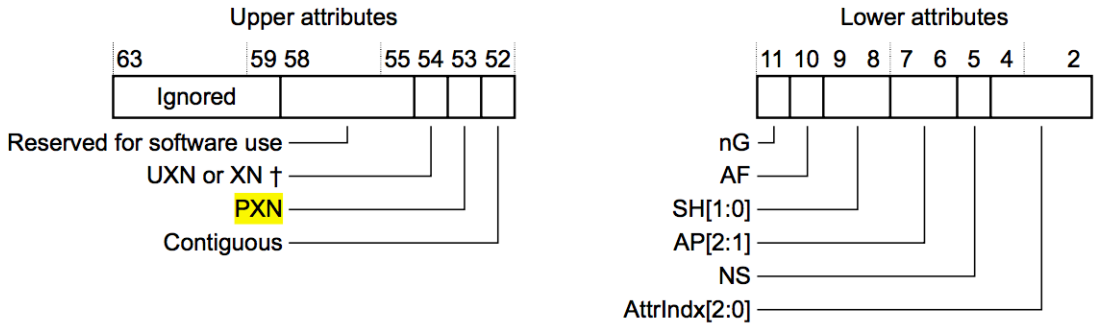
This bit is reserved, SBZ:

- In the EL2 translation regime.
- In the EL3 translation regime.

Attribute fields in stage 1 VMSAv8-64 Block and Page descriptors

In Block and Page descriptors, the memory attributes are split into an upper block and a lower block, as sh
a stage 1
translation:

Attribute fields for VMSAv8-64 stage 1 Block and Page descriptors



Kernel page and page table protection

- rkp_call() handles :
  - allocations, de-allocations of page table
  - manipulations of page entries
- Neither kernel or user space can change attributes of protected pages unauthorizedly
- Related functions
  - pdg_alloc/free()
  - set_pte/pmd/pud()

```
static inline void set_pte(pte_t *ptep, pte_t pte)
{
#ifdef CONFIG_TIMA_RKP
    if (pte && rkp_is_pg_dbl_mapped((u64)(pte))){
        panic("TIMA RKP : Double mapping Detected pte =
                return;
        }
    if (rkp_is_pte_protected((u64)ptep)) {
        rkp_flush_cache((u64)ptep);
        rkp_call(RKP_PTE_SET, (unsigned long)ptep, pte_v
        rkp_flush_cache((u64)ptep);
    } else {
        asm volatile(
            "mov x1, %0\n"
            "mov x2, %1\n"
            "str x2, [x1]\n"
            :
            : "r" (ptep), "r" (pte)
            : "x1", "x2", "memory" );
    }
#else
    *ptep = pte;
#endif /* CONFIG_TIMA_RKP */
```

Kernel data protection

- Based on read only pages
- Read-only global variables
  - RO after initialization
- RKP_RO_AREA located in page ___rkp_ro_start[]
  - struct cred init_cred
  - struct task_secrity_struct init_sec
  - struct security_oprations security_ops

```
#define RKP_RO_AREA __attribute__((section (".rkp.prot.page")))
extern int rkp_cred_enable;
extern char __rkp_ro_start[], __rkp_ro_end[];

extern struct cred init_cred;
```

Tencent

KEEN security lab

Kernel object protection

- Allocated in Read-only pages
  - Writable for hypervisor or TrustZone
- Protected Object type (name of its kmem_cache):
  - cred_jar_ro : credential of processes
  - tsec_jar: security context
  - vfsmnt_cache: struct vfsmount
- Allocation, deallocation and overwriting routines will
  - call rpk_call() to operate read-only objects
- Prevent kernel/user mode manipulating credentials, security context and mount namespace

Tencent

KEEN security lab

History: bypassing trick on S6

- Kernel Object protection had been applied on S6
- Could be bypassed by calling rkp_override_creds()
  - able to override current process’s credentials via rkp_call() in secure world
- Not working on S7
  - S7 add more checking in secure world

- On S6, attacker can call this function to bypass previous kernel object protection

Case study: rkp_override_creds( )

```
#ifdef CONFIG_RKP_KDP
const struct cred *rkp_override_creds(struct cred **cnew)
#else
const struct cred *override_creds(const struct cred *new)
#endif  /* CONFIG_RKP_KDP */
{
    const struct cred *old = current->cred;
#ifdef CONFIG_RKP_KDP
    struct cred *new = *cnew;
    struct cred *new_ro;
    volatile unsigned int rkp_use_count = rkp_get_usecount(new);
    void *use_cnt_ptr = NULL;
    void *tsec = NULL;
#endif  /* CONFIG_RKP_KDP */

    kdebug("override_creds(%p{%d,%d})", new,
           atomic_read(&new->usage),
           read_cred_subscribers(new));

    validate_creds(old);
    validate_creds(new);
#ifdef CONFIG_RKP_KDP
    if(rkp_cred_enable) {
        cred_param_t cred_param;
        new_ro = kmem_cache_alloc(cred_jar_ro, GFP_KERNEL);
        if (!new_ro)
            panic("override_creds(): kmem_cache_alloc() failed");

        use_cnt_ptr = kmem_cache_alloc(usecnt_jar,GFP_KERNEL);
        if(!use_cnt_ptr)
            panic("override_creds() : Unable to allocate usage pointer\n");

        tsec = kmem_cache_alloc(tsec_jar, GFP_KERNEL);
        if(!tsec)
            panic("override_creds() : Unable to allocate security pointer\n");

        rkp_cred_fill_params(new,new_ro,use_cnt_ptr,tsec,RKP_CMD_OVRD_CREDS,rkp_use_count);
        rkp_call(RKP_CMDID(0x46),(unsigned long long)&cred_param,0,0,0,0);

        rocred_uc_set(new_ro,2);
        rcu_assign_pointer(current->cred, new_ro);
```

- To override process's credentials
- Allocate new cred from RO kmem_cache
- Ask RKP to update current cred and security context

Tencent

KEEN security lab

Further cred verifying in secure world

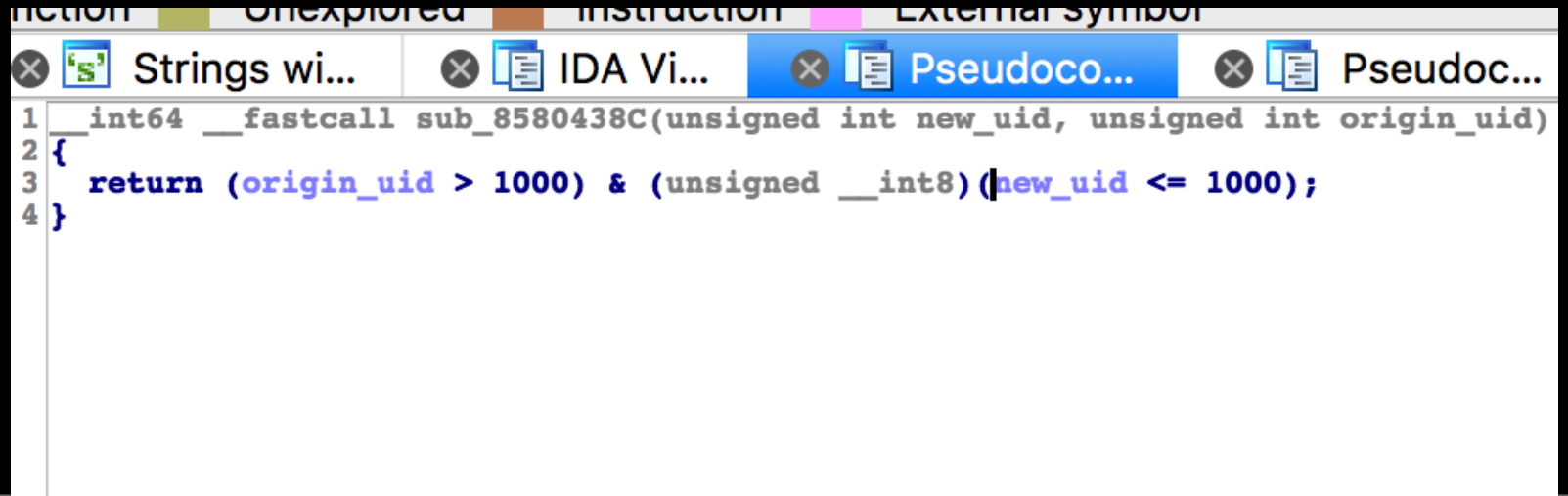
- rpk_override_creds()
  - -> rkp_call(RPK_CMD(0x41)) -> rkp_assign_creds()
- rkp_assign_creds()
  - Real implementation of override_cred() in secure world
  - Additional verifying in KNOX 2.6

```
if ( !v1 )
  return rkp_printk("NULL pData", 0LL, 0LL, 0LL);
__memcpy((__int64)&v11, v1, 0x30uLL);
result = intergrity_chk();
if ( result )
{
  v3 = get_caller_thread_info();
  pcb = get_physical_addr(v3);
  old_cred = get_physical_addr(*(_QWORD *)(pcb + 8 * hardcode_table[17]));
  new_cred = get_physical_addr(v11);
  v7 = get_physical_addr(v12);
  new_cred_copy = v7;
  if ( new_cred && v7 && !sub_85803838(v7) )
  {
    if ( (unsigned int)check_there_is_adbd_zygote(pcb, old_cred) && (unsigned int)uid_checking(new_cred, old_cred) )
    {
      rkp_printk(
        "Priv Escalation!",
        new_cred,
        *(_QWORD *)(new_cred + 8LL * HIDWORD(hardcode_table[19])),
        *(_QWORD *)(old_cred + 8LL * HIDWORD(hardcode_table[19])));
      result = priv_escalation_abort(new_cred, old_cred, 1LL);
    }
```

- Part of Data flow integrity
- UID checking

uid_checking( )

- Check if adbd and zygote has started up
  - If not, allow the override
  - If true, the Android initialize has been finished, start UID checking
- Unprivileged process(uid>1000) cannot override the credential with high privilege(uid 0~1000)
- But still can change its kernel capabilities (very important!)



integrity_checking( )

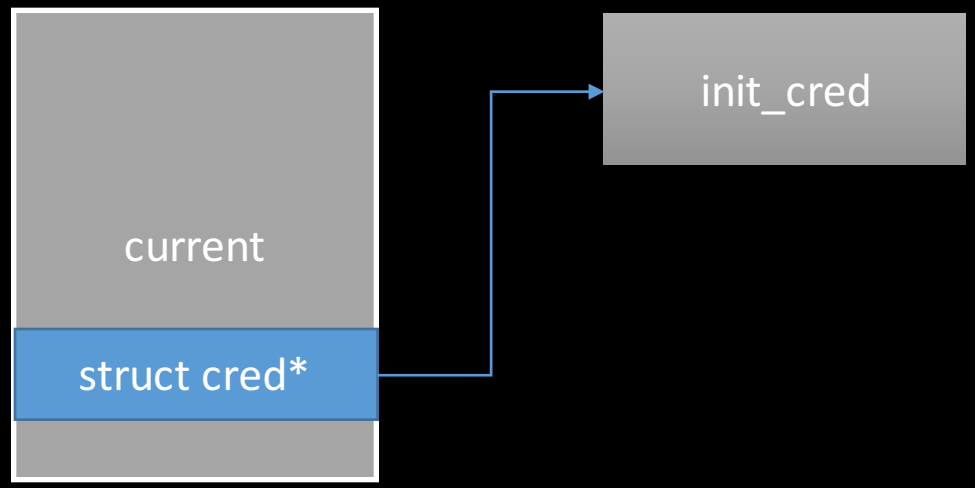
- Do similar checking with security_integrity_current() in Linux Kernel
- Will analyze security_integrity_current() later

Tencent

KEEN security lab

Another old trick to change credential

- For now we know credentials are READ-ONLY
- What if we reuse init's credential?



- Not working on S7,because of Data Flow Integrity

Data Flow Integrity

- New in KNOX 2.6
- Implemented in both Linux kernel and Secure world
  - security_integrity_current() (kernel)
  - Integrity_checking()
- Additional members in struct cred{}

```
#endif
    struct user_struct *user;    /
    struct user_namespace *user_n
    struct group_info *group_info
    struct rcu_head rcu;         /
#ifdef CONFIG_RKP_KDP
    atomic_t *use_cnt;
    struct task_struct *bp_task;
    void *bp_pgd;
    unsigned long long type;
#endif /*CONFIG_RKP_KDP*/
};
#ifdef CONFIG_RKP_KDP
```

- use_cnt: pointer to refcount of cred
- bp_task: pointer to this cred's owner
- bp_pgd: pointer to process's PGD
- type: not used in DFI

Data Flow Integrity

- New in KNOX 2.6
- Implemented in both Linux kernel and Secure world
  - security_integrity_current() (kernel)
  - Integrity_checking()
- Additional members in struct task_security_struct{}

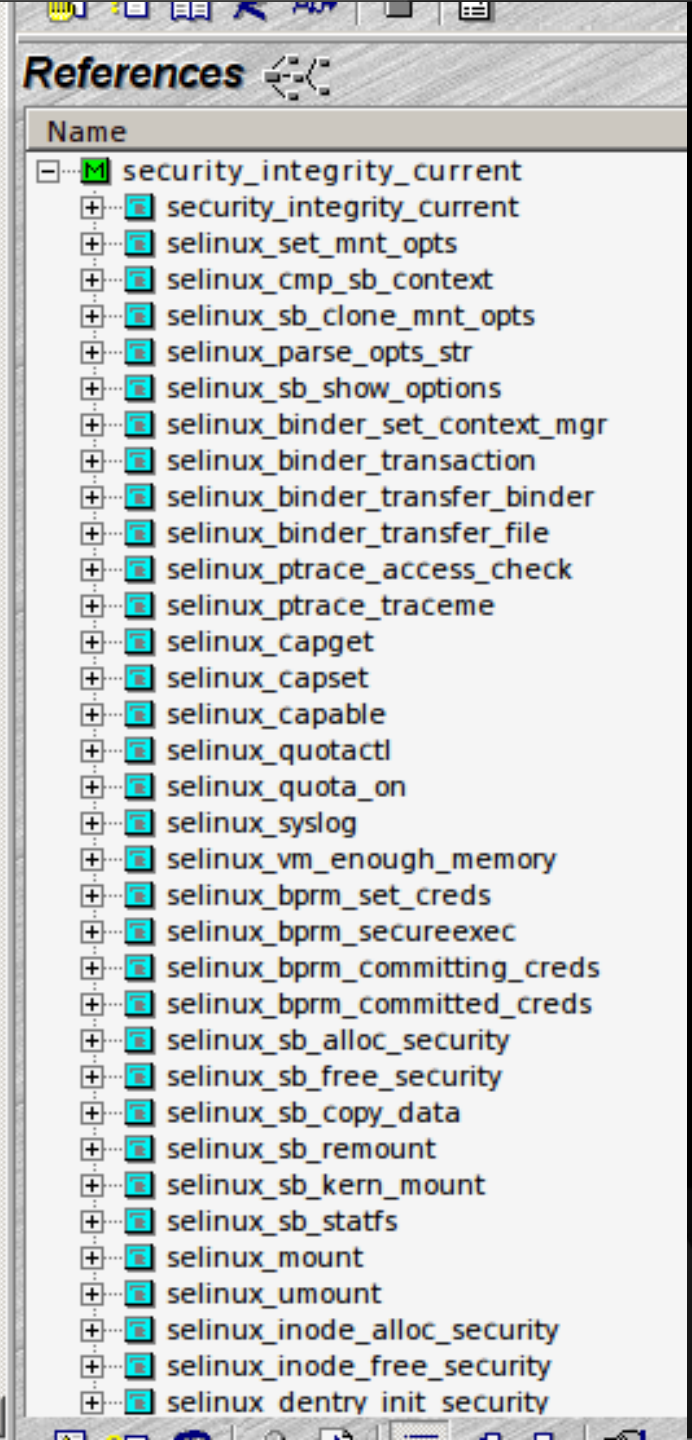
```
struct task_security_struct {
    u32 osid;       /* SID prior to last exe
    u32 sid;        /* current SID */
    u32 exec_sid;       /* exec SID */
    u32 create_sid;     /* fscreate SID */
    u32 keycreate_sid;  /* keycreate SID */
    u32 sockcreate_sid; /* fscreate SID */
#ifdef CONFIG_RKP_KDP
    void *bp_cred;
#endif
};
```

- bp_cred: pointer to this context's owner cred

security_integrity_current( )

- Hard-coded hooking in every SELinux routines
  - Verify process's credential in real-time
  - To check if
    - current *struct cred{}* and *struct task_security_struct{}* are allocated in RO page
    - cred->bp_task is current process's
    - task_security->bp_cred is current cred
    - current mount namespace is malformed

```
}
/* Main function to verify cred security context of a process */
int security_integrity_current(void)
{
    if ( rkp_cred_enable &&
         (rkp_is_valid_cred_sp((u64)current_cred(),(u64)current_cred()->secu
         cmp_sec_integrity(current_cred(),current->mm)||
         cmp_ns_integrity())) {
         rkp_print_debug();
         panic("RKP CRED PROTECTION VIOLATION\n");
    }
    return 0;
}
unsigned int rkp_get_task_sec_size(void)
```



Summary of RKP and DFI

- Even we achieved arbitrary kernel memory overwriting, we cannot:
  - Manipulate credentials and security context in kernel mode
  - Point current credential to init_cred
  - Call rkp_override_creds() to ask secure world to help us override credential with uid 0~1000
- But we still can:
  - Call kernel function from user mode
    - Hijacking *ptmx_fops->check_flags(int flag)*
    - The number of parameters is limited
    - Only low 32bit of X0 is controllable
  - Override credential with full kernel capabilities (cred->cap_**)
  - Overwrite unprotected data in kernel

Tencent

KEEN security lab

Bypassing RKP and DFI

- Main idea: ask kernel to create a privileged process for me
- Creating a root process
- I can’t call *call_usermodehelper(path,argv,envp,wait)* via *ptmx_fops->check_flags(flag)*
- Call orderly_poweroff() instead

orderly_poweroff()

Tencent

```c
/**
 * orderly_poweroff - Trigger an orderly system poweroff
 * @force: force poweroff if command execution fails
 *
 * This may be called from any context to trigger a system shutdown.
 * If the orderly shutdown fails, it will force an immediate shutdown
 */
int orderly_poweroff(bool force)
{
    if (force) /* do not override the pending "true" */
        poweroff_force = true;
    schedule_work(&poweroff_work);
    return 0;
}
EXPORT_SYMBOL_GPL(orderly_poweroff);
```

- Call ___oderly_poweroff() in worker thread

```c
char poweroff_cmd[POWEROFF_CMD_PATH_LEN] = "/sbin/poweroff";

static int __orderly_poweroff(bool force)
{
    char **argv;
    static char *envp[] = {
        "HOME=/",
        "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
        NULL
    };
    int ret;

    argv = argv_split(GFP_KERNEL, poweroff_cmd, NULL);
    if (argv) {
        ret = call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
        argv_free(argv);
    } else {
```

- Worker create a new root process, cmd is *poweroff_cmd*
- *poweroff_cmd* is writeable

KEEN security lab

Bypassing steps

- Call rpk_override_creds() via ptmx_fops->check_flags()
  - Override own cred to gain full kernel capabilities
  - But don’t change uid
- Overwrite *poweroff_cmd* with “/data/data/***/ss7kiler”
- Call orderly_poweroff() via ptmx_fops->check_flags()
- Modify ss7killer’s thread_info->address limit
- ss7killer: call rpk_override_creds() to change its sid from *u:r:kernel:s0* to *u:r:init:s0*

Result: privileged ss7killer

Tencent

- root
- u:r:init:s0

```
shell@hero2qltechn:/ $ ps -Z | grep init
u:r:init:s0                  root      1     0     /init
u:r:init:s0                  root      20592 1     /data/local/tmp/ss7killer
u:r:init:s0                  root      20608 20592 su
u:r:init:s0                  root      20611 1     daemonsu:mount:master
u:r:init:s0                  root      20614 1     daemonsu:master
u:r:init:s0                  root      20797 20614 daemonsu:0
u:r:init:s0                  root      21161 20614 daemonsu:10193
u:r:init:s0                  root      21163 21161 daemonsu:10193:21157
```

KEEN security lab

u:r:init:s0

- Not good enough
- Still be limited by SELinux
- Almost can do nothing…
- Disabling/Bypassing SELinux is necessary

Tencent

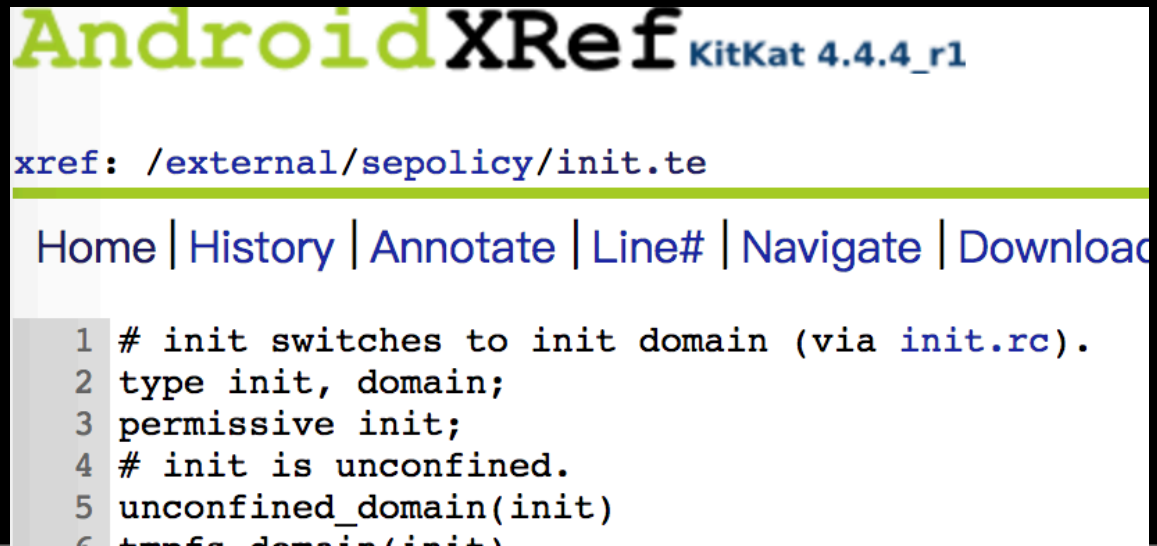
KEEN security lab

SELinux enhancement

- Disabled CONFIG_SECURITY_SELINUX_DEVELOP long time ago
  - Cannot disable SELinux by overwrite selinux_enforcing
  - Statically enforcing all the time
- init process cannot reload SELinux policy after system initialized
- Permissive domain is not allowed

Permissive domain

- Officially used by Google before Lollipop
  - For policy developing purpose
- All domains are non-permissive since Lollipop
- Domains still can be switched to permissive mode by policy reloading (/sys/fs/selinux/load)



Permissive domain – kernel support

- A permissive domain's access vector decision(AVD) will be set AVD_FLAGS_PERMISSIVE
- All operations are permitted

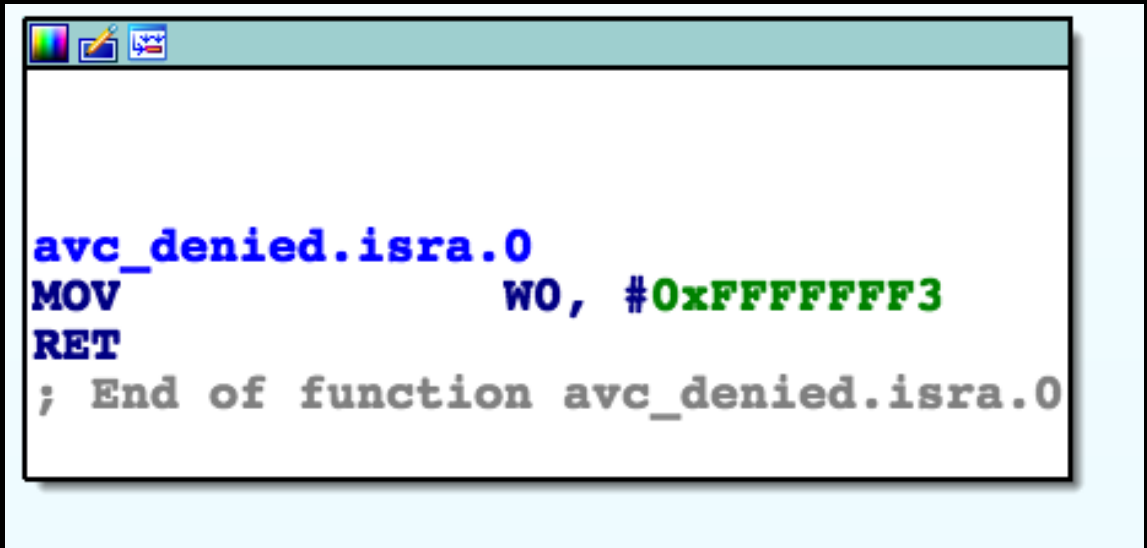
```
static noinline int avc_denied(u32 ssid, u32 tsid,
                u16 tclass, u32 requested,
                u8 driver, u8 xperm, unsigned flags,
                struct av_decision *avd)
{
    if (flags & AVC_STRICT)
        return -EACCES;

    if (selinux_enforcing && !(avd->flags & AVD_FLAGS_PERMISSIVE))
        return -EACCES;

    avc_update_node(AVC_CALLBACK_GRANT, requested, driver, xperm, ssid,
                tsid, tclass, avd->seqno, NULL, flags);
    return 0;
}
```

S7 removed AVD_FLAGS_PERMISSIVE

- avc_denied always simply return –EACCES

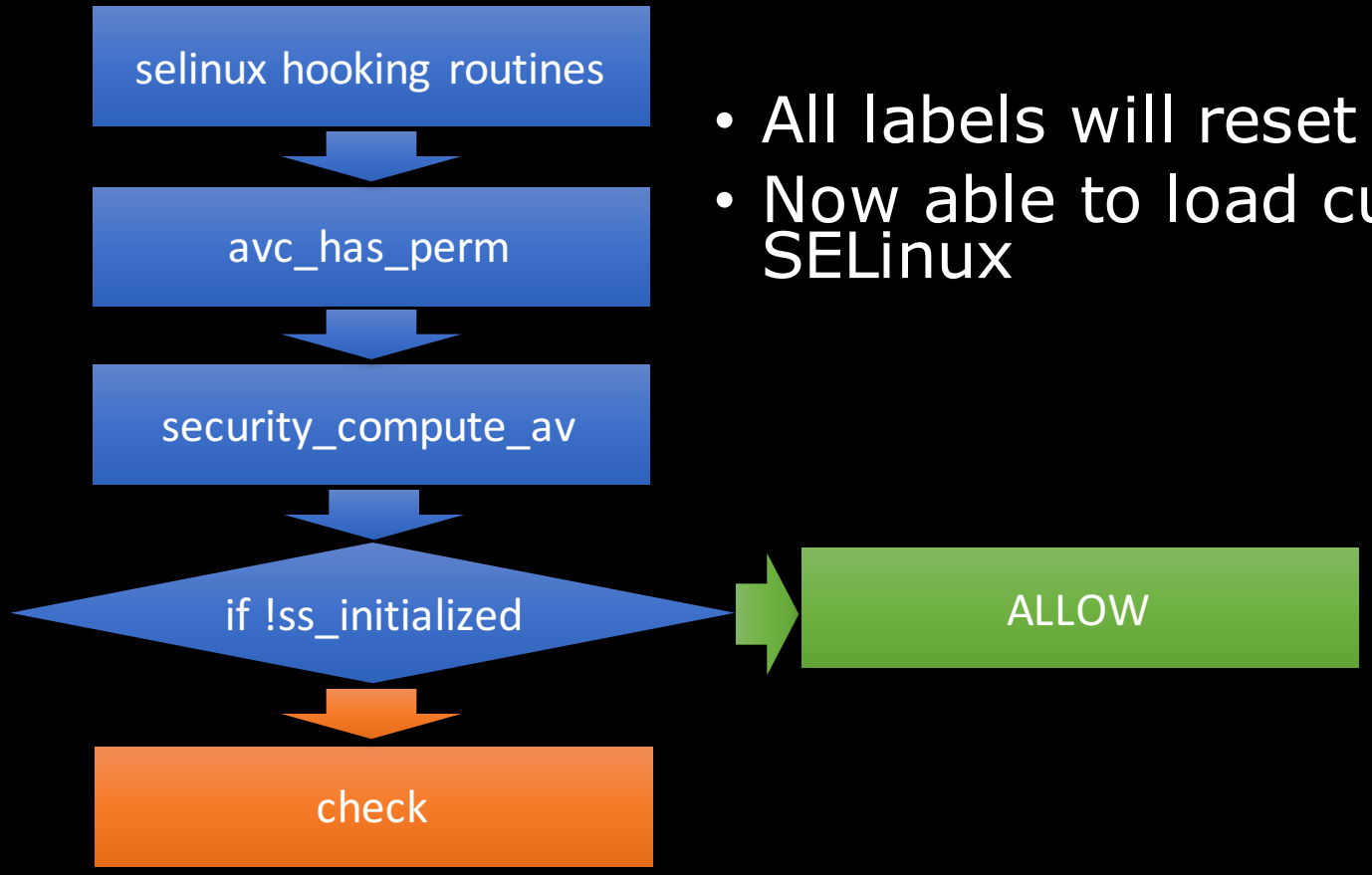


```
avc_denied.isra.0
MOV             W0, #0xFFFFFFF3
RET
; End of function avc_denied.isra.0
```

Bypass SELinux on S7

Tencent

- Cheating kernel that SELinux is not initialized yet
  - Depends on global variable ss_initialized (writable)



- All labels will reset to none except kernel domain
- Now able to load customized policy and reinitialize SELinux

KEEN security lab

After setting ss_initialized = 0

- All labels missed except kernel
  - SELinux must be re-enabled ASAP, or Apps may corrupt files' label permanently
- Load customized policy and reinitialize SELinux

```
kernel          root       900   2     dhd_watchdog_th
kernel          root       901   2     dhd_rpm_state_t
-               system     903   1     /system/bin/factory.adsp
-               net_admin  910   1     /system/bin/ipacm
-               radio      914   1     /system/vendor/bin/qti
-               radio      922   1     /system/bin/netmgrd
-               radio      947   1     /system/bin/rild
-               jack       950   694   androidshmservice
kernel          root       1157  2     irq/140-arm-smm
kernel          root       1160  2     irq/141-arm-smm
-               system     1161  1     /system/bin/mcDriverDaemon
kernel          root       1209  2     tee_scheduler
kernel          root       1218  2     irq/162-arm-smm
kernel          root       1219  2     irq/167-arm-smm
kernel          root       1221  2     irq/163-arm-smm
kernel          root       1223  2     irq/168-arm-smm
-               system     1250  728   system_server
```

Policy customizations

- Policy database locate at /sys/fs/selinux/policy
- Modify the database with libsepol API
  - Load policy DB to the user memory
  - Add rules into database
    - Allow untrusted_app, init, toolbox domain to do everything
  - Ask kernel to reload the database
- Set ss_initialized to 1

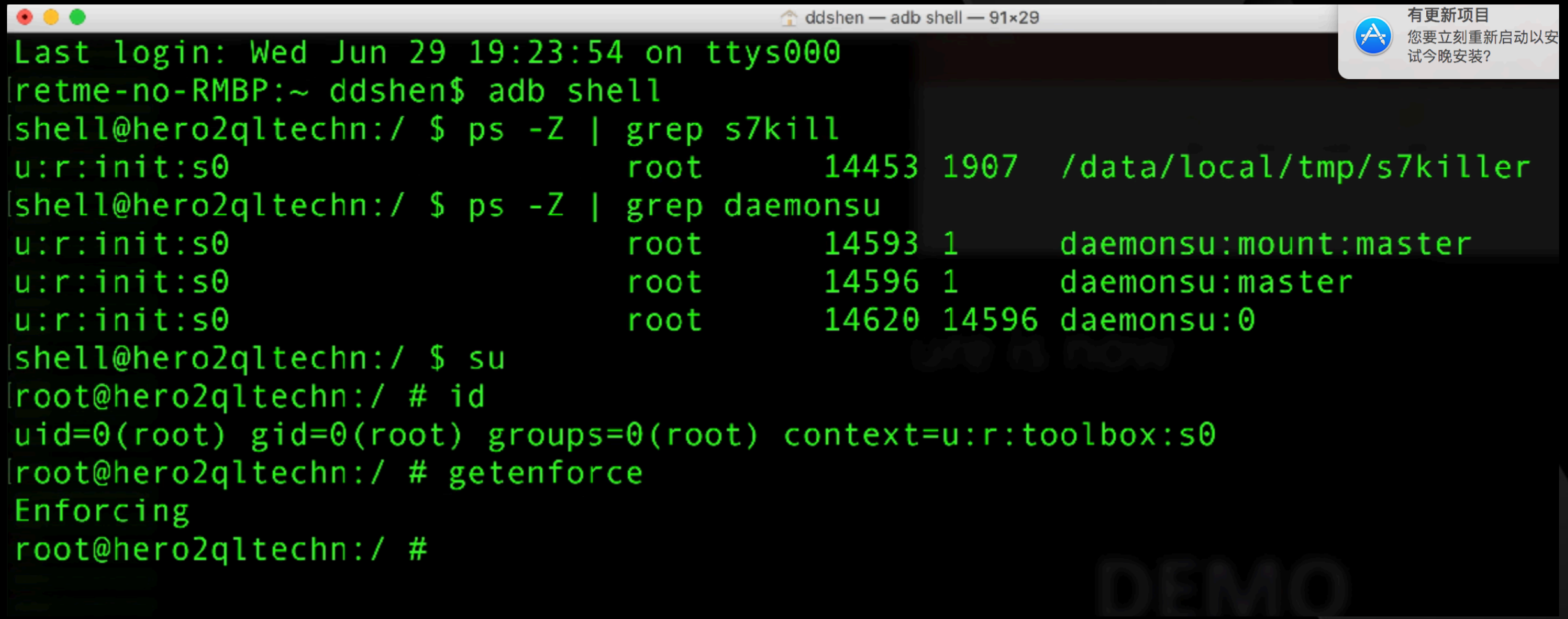
Tencent

KEEN security lab

Gain Root

- Leaking kernel information √
- Bypassing KASLR √
- Overwriting arbitrary memory √
- Bypassing RKP & DFI √
- Bypassing enforced SELinux √

Tencent



Tencent

KEEN
security
lab