Time Bomb : Universal Root Is Coming!

Tong Lin, Jun Yao

C0RE Team

About us & C0RE Team

- Tong Lin & Jun Yao
  - Security researcher and developer @C0RE Team
  - Focus on kernel exploit discover

- C0RE Team(c0reteam.org)
  - A security-focused group started in mid-2015, with a recent focus on the Android/Linux platform
  - Aim to hunt and exploit zero-day vulnerabilities
  - 131 public CVEs for AOSP and Linux Kernel currently
  - Google Android Security Top Researcher Team

Agenda

- Introduction of timerfd
- Vulnerability analysis
- Exploitation
- Conclusion

Introduction of timerfd

- A timer interface provided by Linux for the user program.
- Based on file descriptors which allow timer event to be used with standard POSIX poll(), select() and read().
- Three main system calls(timerfd_create(), timerfd_settime(),timerfd_gettime()) associated with it.

```
SYSCALL_DEFINE2(timerfd_create, int, clockid, int, flags)

SYSCALL_DEFINE4(timerfd_settime, int, ufd, int, flags,
        const struct itimerspec __user *, utmr,
        struct itimerspec __user *, otmr)

SYSCALL_DEFINE2(timerfd_gettime, int, ufd, struct itimerspec __user *, otmr)
```

Vulnerability analysis

——Timeline

- CVE-2017-10661
- Discovered by C0RE Team with syzkaller(https://github.com/google/syzkaller) in January 2017
- Reported to Google in March 2017
- Made public in Android Security Bulletin—August 2017

Vulnerability analysis

——Upstream Patch

- The handling of the might_cancel queueing is not properly protected, so parallel operations on the file descriptor can race with each other.
- Protect the context for these operations with a seperate lock.

```
Diffstat
-rw-r--r-- fs/timerfd.c 17
1 files changed, 14 insertions, 3 deletions

diff --git a/fs/timerfd.c b/fs/timerfd.c
index c173cc1..384fa75 100644
--- a/fs/timerfd.c
+++ b/fs/timerfd.c
@@ -40,6 +40,7 @@ struct timerfd_ctx {
 	short unsigned settime_flags;	/* to show in fdinfo */
 	struct rcu_head rcu;
 	struct list_head clist;
+	spinlock_t cancel_lock;
 	bool might_cancel;
 };

@@ -112,7 +113,7 @@ void timerfd_clock_was_set(void)
 	rcu_read_unlock();
 }

-static void timerfd_remove_cancel(struct timerfd_ctx *ctx)
+static void __timerfd_remove_cancel(struct timerfd_ctx *ctx)
 {
 	if (ctx->might_cancel) {
 		ctx->might_cancel = false;
@@ -122,6 +123,13 @@ static void timerfd_remove_cancel(struct timerfd_ctx *ctx)
 	}
 }

+static void timerfd_remove_cancel(struct timerfd_ctx *ctx)
+{
+	spin_lock(&ctx->cancel_lock);
+	__timerfd_remove_cancel(ctx);
+	spin_unlock(&ctx->cancel_lock);
+}
+
 static bool timerfd_canceled(struct timerfd_ctx *ctx)
 {
 	if (!ctx->might_cancel || ctx->moffs != KTIME_MAX)
@@ -132,6 +140,7 @@ static bool timerfd_canceled(struct timerfd_ctx *ctx)

 static void timerfd_setup_cancel(struct timerfd_ctx *ctx, int flags)
 {
+	spin_lock(&ctx->cancel_lock);
 	if ((ctx->clockid == CLOCK_REALTIME ||
 	     ctx->clockid == CLOCK_REALTIME_ALARM) &&
 	    (flags & TFD_TIMER_ABSTIME) && (flags & TFD_TIMER_CANCEL_ON_SET)) {
@@ -141,9 +150,10 @@ static void timerfd_setup_cancel(struct timerfd_ctx *ctx, int flags)
 			list_add_rcu(&ctx->clist, &cancel_list);
 			spin_unlock(&cancel_lock);
 		}
-	} else if (ctx->might_cancel) {
-		timerfd_remove_cancel(ctx);
+	} else {
+		__timerfd_remove_cancel(ctx);
 	}
+	spin_unlock(&ctx->cancel_lock);
 }

 static ktime_t timerfd_get_remaining(struct timerfd_ctx *ctx)
@@ -400,6 +410,7 @@ SYSCALL_DEFINE2(timerfd_create, int, clockid, int, flags)
 		return -ENOMEM;

 	init_waitqueue_head(&ctx->wqh);
+	spin_lock_init(&ctx->cancel_lock);
 	ctx->clockid = clockid;

 	if (isalarm(ctx))
```

Vulnerability analysis

——Details

- Syscall timerfd_settime() will perform list_add and list_del operations on the “struct list_head ctx->clist” through function timerfd_setup_cancel() and timerfd_remove_cancel().

```
timerfd_settime()
        \__ do_timerfd_settime()
                \__ timerfd_setup_cancel()
                        \__ list_add_rcu()

timerfd_settime()
        \__ do_timerfd_settime()
                \__ timerfd_setup_cancel()
                        \__ timerfd_remove_cancel()
                                \__ list_del_rcu()
```

Vulnerability analysis

——Details

```c
static void timerfd_setup_cancel(struct timerfd_ctx *ctx, int flags)
{
	if ((ctx->clockid == CLOCK_REALTIME ||
	     ctx->clockid == CLOCK_REALTIME_ALARM ||
	     ctx->clockid == CLOCK_POWEROFF_ALARM) &&
	    (flags & TFD_TIMER_ABSTIME) && (flags & TFD_TIMER_CANCEL_ON_SET)) {
		if (!ctx->might_cancel) {
			ctx->might_cancel = true;
			spin_lock(&cancel_lock);
			list_add_rcu(&ctx->clist, &cancel_list);
			spin_unlock(&cancel_lock);
		}
	} else if (ctx->might_cancel) {
		timerfd_remove_cancel(ctx);
	}
}
```

- The protection of the might_cancel queueing is by setting “ctx->might_cancel” to true or false.

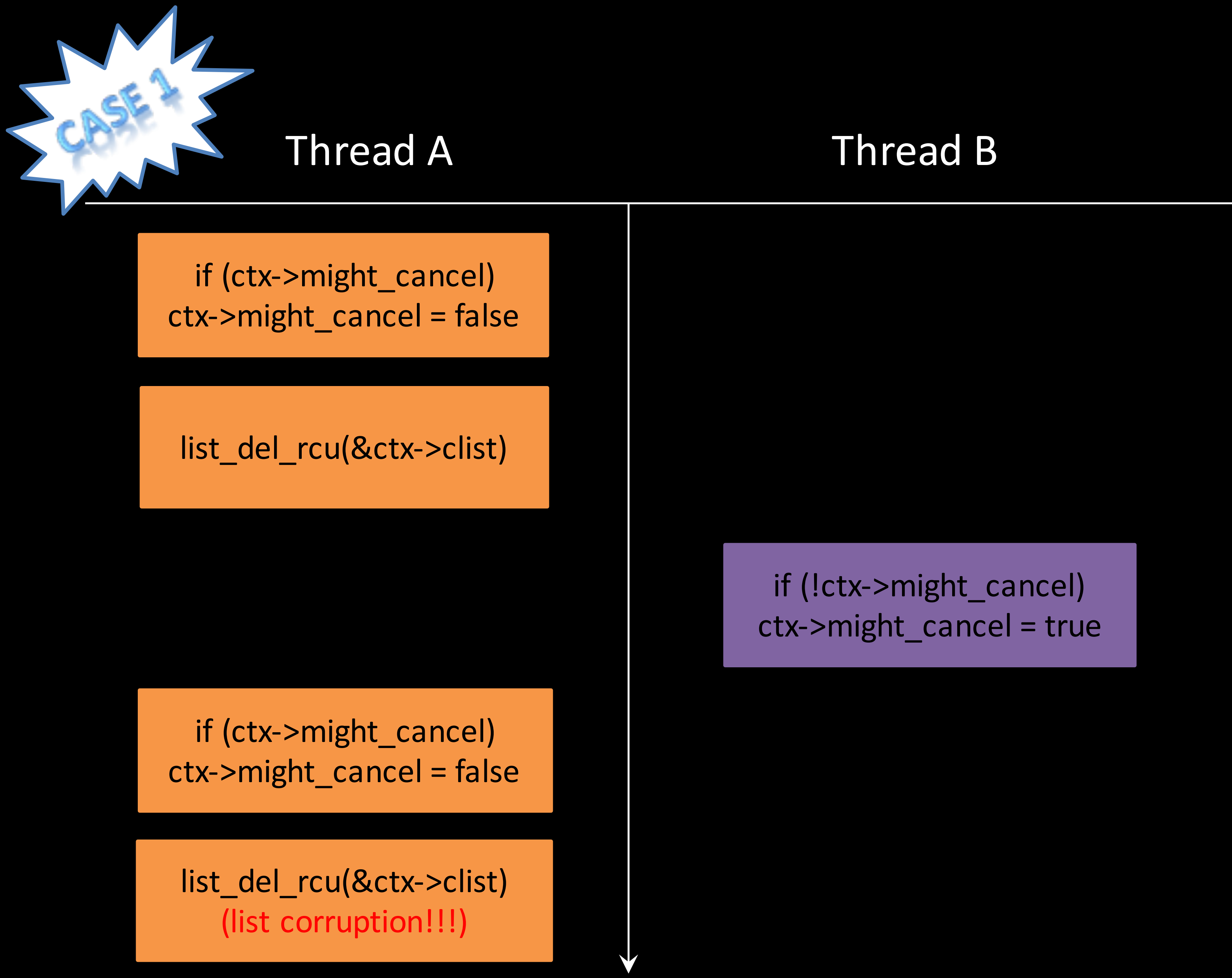
- However, this does not prevent the race condition.
- The parallel handle of the might_cancel queue which may race with each other.

```c
static void timerfd_remove_cancel(struct timerfd_ctx *ctx)
{
	if (ctx->might_cancel) {
		ctx->might_cancel = false;
		spin_lock(&cancel_lock);
		list_del_rcu(&ctx->clist);
		spin_unlock(&cancel_lock);
	}
}
```

Vulnerability analysis

——Result

- Assume that two threads run timerfd_settime() at the same time, what will happen?
- Here look at these two cases.


CASE 1
Thread A
Thread B
if (ctx->might_cancel)
ctx->might_cancel = false
list_del_rcu(&ctx->clist)
if (!ctx->might_cancel)
ctx->might_cancel = true
if (ctx->might_cancel)
ctx->might_cancel = false
list_del_rcu(&ctx->clist)
(list corruption!!!)

CASE 2

Thread A	Thread B
if (!ctx->might_cancel) ctx->might_cancel = true	
list_add_rcu(&ctx->clist, &cancel_list)	
	if (ctx->might_cancel) ctx->might_cancel = false
if (!ctx->might_cancel) ctx->might_cancel = true	
list_add_rcu(&ctx->clist, &cancel_list)	
	list_del_rcu(&ctx->clist) (dangling pointer!!!)

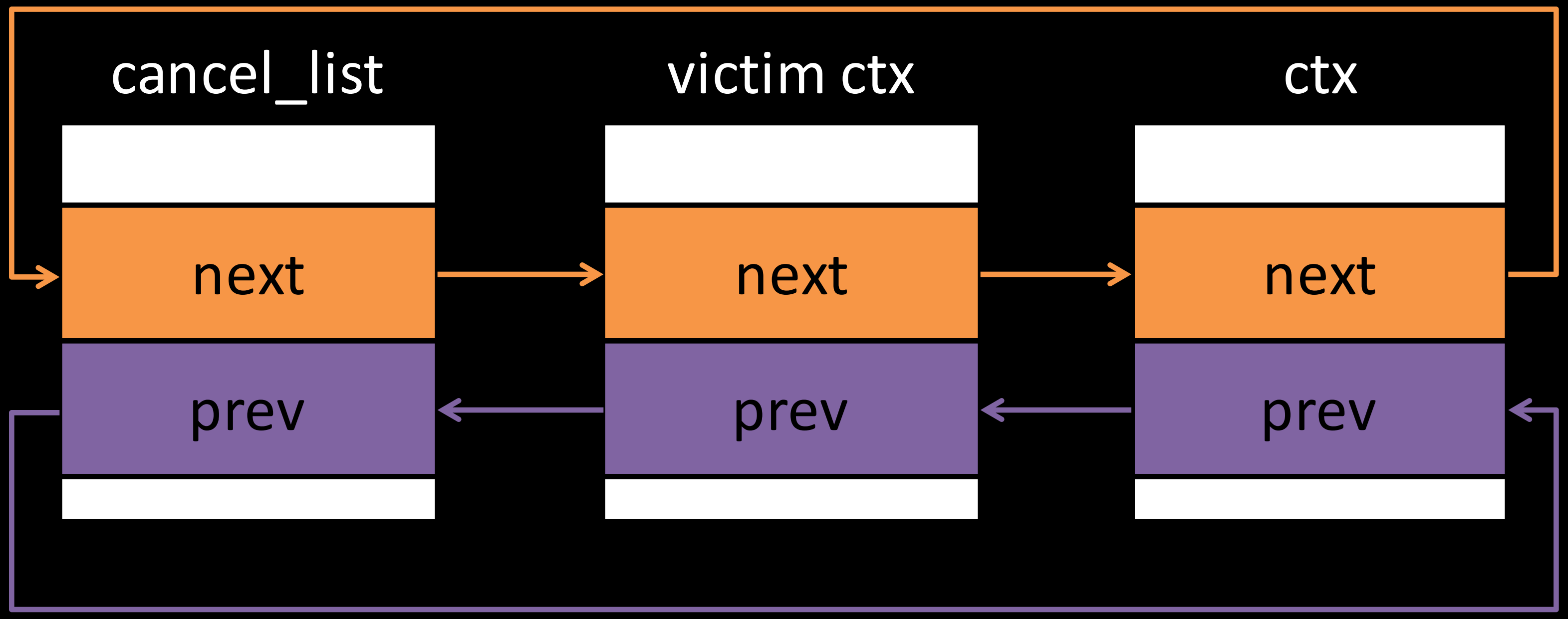
Exploitation

- The exploit chain probably can be divided into six steps.
  - Step 1 : Call timerfd_create() to creat a timerfd and alloc a ctx.
  - Step 2 : Race timerfd_settime() to list_add “ctx->clist” twice.
  - Step 3 : Close(fd) to kfree ctx.
  - Step 4 : Fill in the victim SLAB (Heap spray).
  - Step 5 : Trigger function pointer to control “PC”.
  - Step 6 : Using gadget for JOP to modify address_limit and defeat PXN.

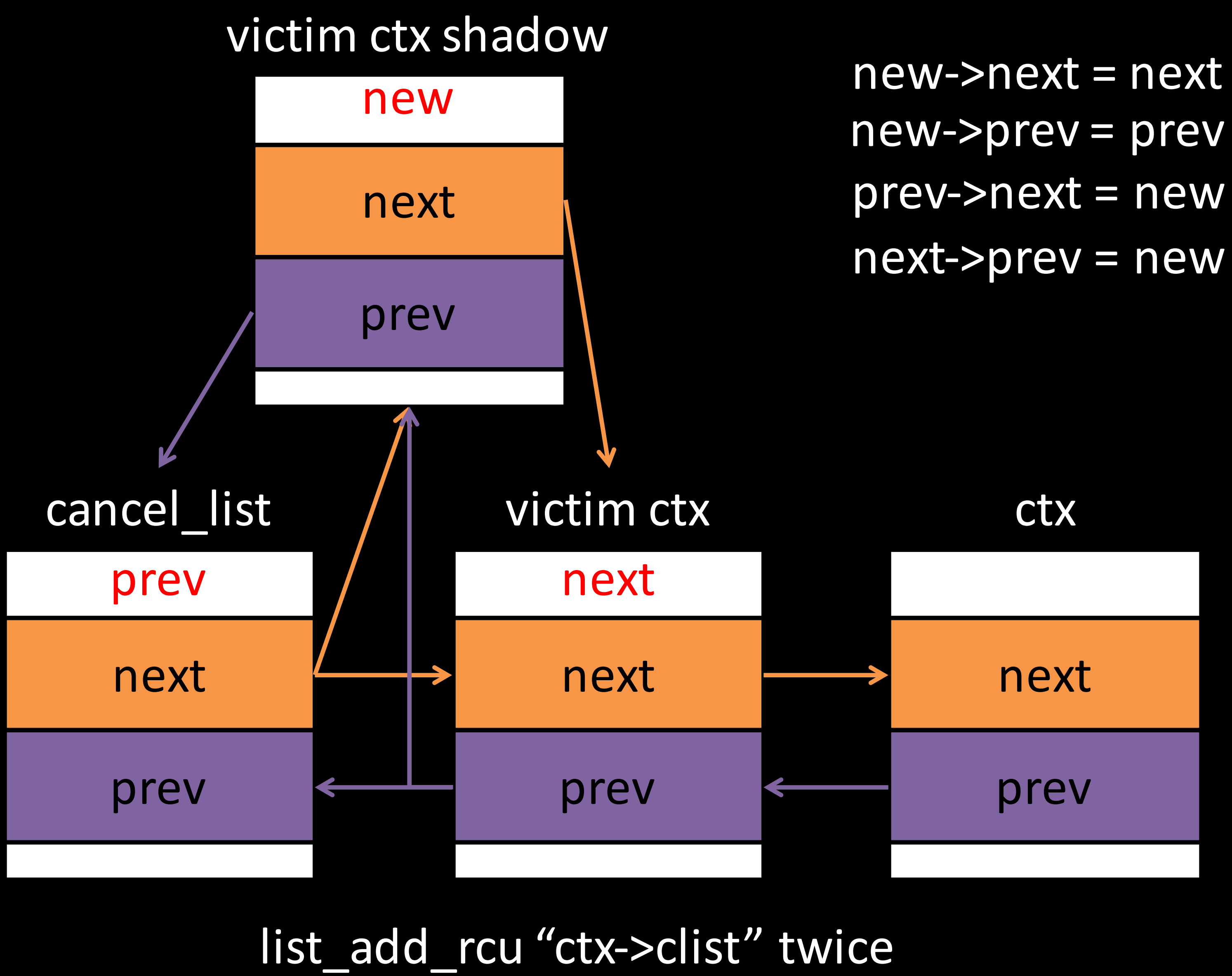
Exploitation

——Details of step 2

timerfd_settime() --> do_timerfd_settime()
--> timerfd_setup_cancel()

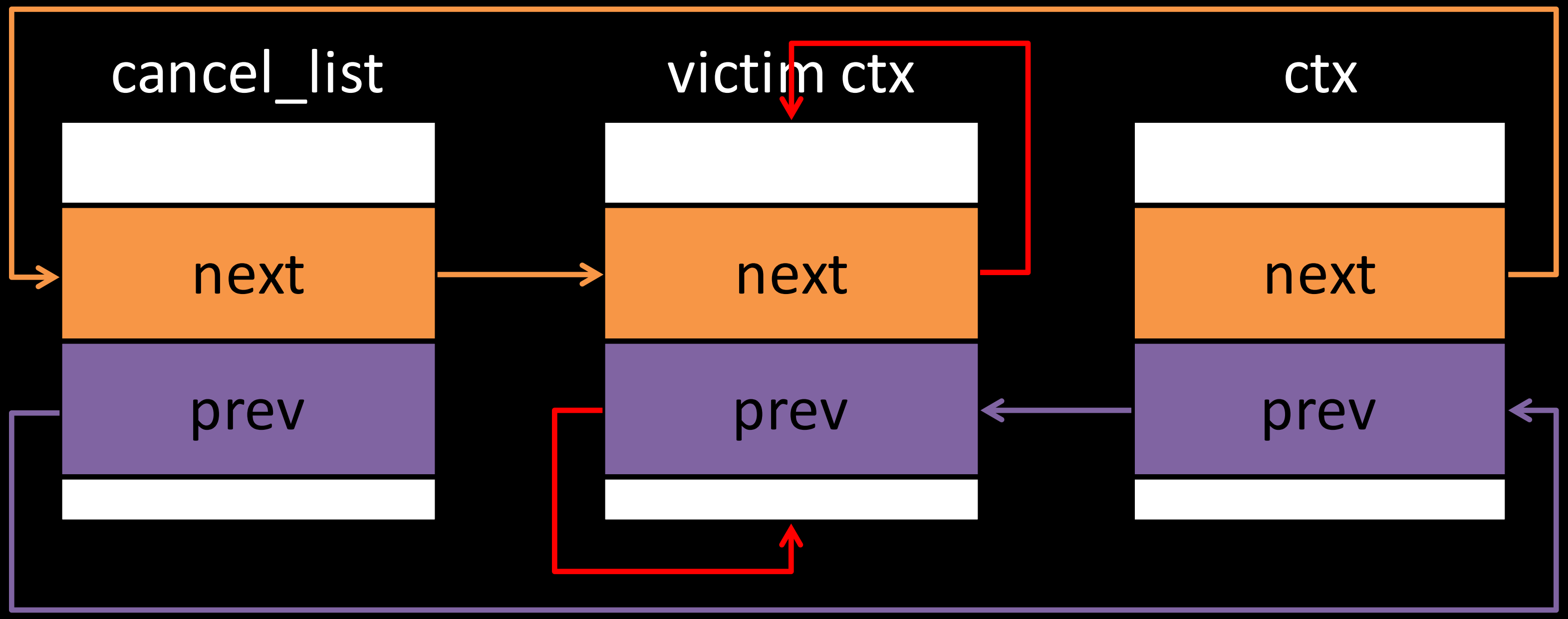


list_add_rcu “ctx->clist”


victim ctx shadow
new
next
prev
new->next = next
new->prev = prev
prev->next = new
next->prev = new
cancel_list
prev
next
prev
victim ctx
next
next
prev
ctx
next
prev
list_add_rcu “ctx->clist” twice

Exploitation

——Details of step 2

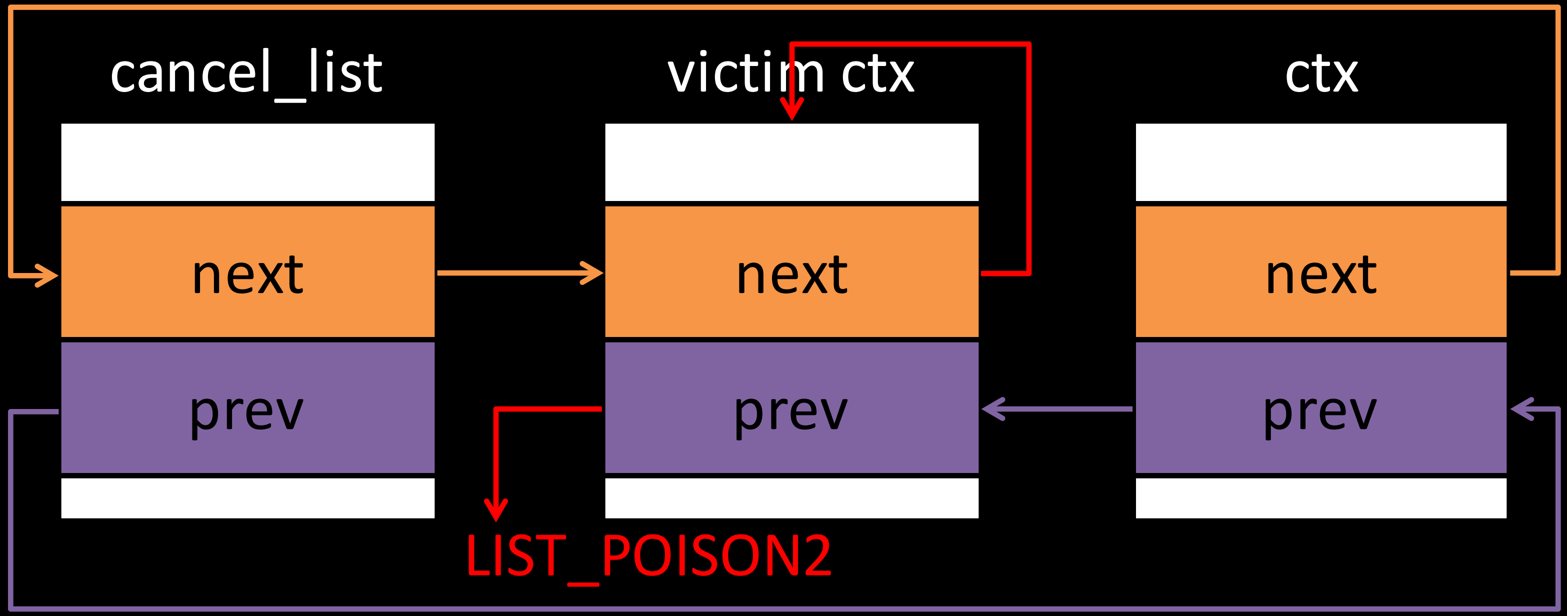


list_add_rcu “ctx->clist” twice thrice ...

Exploitation

——Details of step 3

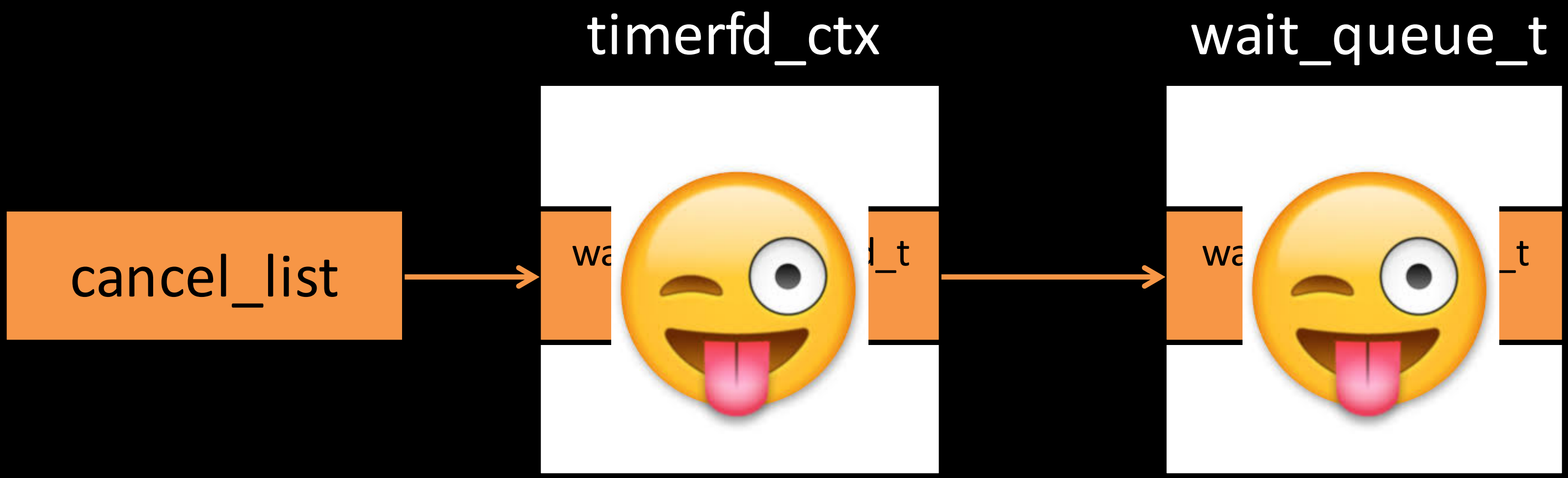
close() --> timerfd_release()
                    |--> timerfd_remove_cancel(ctx)
                    |--> kfree_rcu(ctx)



list_del_rcu “ctx->clist”

Exploitation

——Details of step 4



timerfd_clock_was_set() --> wake_up_locked()
--> __wake_up_common(ctx)

Exploitation

——Details of step 5

- Trigger “curr->func” function pointer to control “PC”
- This process can be achieved by combining other AOSP vulnerability

Conclusion

- In recent years, Google has been committed to improving the security of the Android ecosystem.
- A number of mechanisms within Android was enabled, including memory protection, attack surface reduction, selinux enhancement and so on.
- As a result, to find a universal root solution for Android is becoming more and more challenging.
- So development of new vulnerability digging tools and accumulation of stable vulnerability exploit techniques are very necessary.

Q&A

CVE-2017-0326, CVE-2017-0709, CVE-2017-0739, CVE-2017-0737, CVE-2017-0731, CVE-2017-10661,
CVE-2017-8264,CVE-2017-0684,CVE-2017-0666,CVE-2017-0681,CVE-2017-0665,CVE-2017-0651,CVE-2017-7368,
CVE-2017-0564,CVE-2017-0483,CVE-2017-0526,CVE-2017-0527,CVE-2017-0333,CVE-2017-0479,CVE-2017-0480,
CVE-2017-0450,CVE-2017-0448,CVE-2017-0436,CVE-2017-0444,CVE-2017-0435,CVE-2017-0429,CVE-2017-0428,
CVE-2017-0425,CVE-2017-0418,CVE-2017-0417,CVE-2017-0402,CVE-2017-0401,CVE-2017-0400,CVE-2017-0398,
CVE-2017-0385,CVE-2017-0384,CVE-2017-0383,CVE-2016-10291,CVE-2016-8481,CVE-2016-8480,CVE-2016-8449,
CVE-2016-8435,CVE-2016-8432,CVE-2016-8431,CVE-2016-8426,CVE-2016-8425,CVE-2016-8400,CVE-2016-8392,
CVE-2016-8391,CVE-2016-6791,CVE-2016-6790,CVE-2016-6789,CVE-2016-6786,CVE-2016-6780,CVE-2016-6777,
CVE-2016-6775,CVE-2016-6765,CVE-2016-6761,CVE-2016-6760,CVE-2016-6759,CVE-2016-6758,CVE-2016-6746,
CVE-2016-6736,CVE-2016-6735,CVE-2016-6734,CVE-2016-6733,CVE-2016-6732,CVE-2016-6731,CVE-2016-6730,
CVE-2016-6720,CVE-2016-3933,CVE-2016-3932,CVE-2016-3909,CVE-2016-5342,CVE-2016-3895,CVE-2016-3872,
CVE-2016-3871,CVE-2016-3870,CVE-2016-3857,CVE-2016-3844,CVE-2016-3835,CVE-2016-3825,CVE-2016-3824,
CVE-2016-3823,CVE-2016-3774,CVE-2016-3773,CVE-2016-3772,CVE-2016-3771,CVE-2016-3770,CVE-2016-3765,
CVE-2016-3747,CVE-2016-3746,CVE-2016-2486,CVE-2016-2485,CVE-2016-2484,CVE-2016-2483,CVE-2016-2482,
CVE-2016-2481,CVE-2016-2480,CVE-2016-2479,CVE-2016-2478,CVE-2016-2477,CVE-2016-2452,CVE-2016-2451,
CVE-2016-2450,CVE-2016-2449,CVE-2016-2448,CVE-2016-2442,CVE-2016-2441,CVE-2016-2437,SVE-2016-5393,
CVE-2015-1805,CVE-2016-0826,CVE-2016-0804,CVE-2015-8681,CVE-2015-8318,CVE-2015-8307,CVE-2015-5524,
CVE-2015-8089,CVE-2015-3869,CVE-2015-3868,CVE-2015-3865,CVE-2015-3862,CVE-2015-0573,CVE-2015-0568