★ 看雪社区 > Android安全 发新帖









ncore bypass)

▲ 举报

② 87

重发调整了格式

syscall 入口

38分钟前

所有的参数都是syscall传进内核区分参数,指令

参数:设置/移除pid,设置/移除文件过滤字符串设置/移除 nofity bypass文件路径

[原创]【从源码过反调试】三、源码实现一些过检测(文件过滤,进程隐藏,notify bypass,mi

指令: 开关文件过滤, 物理地址读写内存, 进程隐藏开关。nofity文件过滤

```
enum CmdDataType
1
2
3
4
        SET_TARGET_PID = 1 ,
5
6
7
        SET_SELF_PID = 2,
8
9
        REMOVE\_TARGET\_PID = 4,
10
        REMOVE_SELF_PID = 8,
11
12
        SET_FILTER_STR = 16,
13
14
15
        REMOVE_FILTER_STR = 32,
16
17
        SET_NOTIFY_PATH_STR = 64,
18
        REMVOE_NOTIFY_PATH_STR = 128
19
20
    };
21
22
23
24
25
    enum CmdSwitchType
26
27
28
        FILTER_FILE = 0,
29
30
31
        GLOABAL_PID,
32
33
        READ_MEMORY,
34
        WRITE_MEMORY,
35
36
        HIDE_PROCESS,
37
38
         REMOVE_HIDE_PROCESS,
39
40
        NOTIFY
41
42
43 };
```

目标管理

用户层传参数、指令,保存到内核。

开启各种功能最先做的还是选定目标了,设置target_pid开启各种过检测,设置spid保护自己的进程

文件过滤

过通用文件检测,主要用来过一些模拟器检查。可以自定义返回错误码









在getname_flags里增加判断,我测试过几个点,这个位置是文件访问的几个syscall(open, access, stat) 都会走的,在这里比较合适,不过不知道有没有遗漏其他的。

修改源码fs/namei.c

判断要访问的文件是否在被保护字符串列表里。

```
1
    struct filename *
2
3
    getname_flags(const char __user *filename, int flags, int *empty)
4
5
6
7
8
            // ... 前面省略
9
10
11
12
        /* The empty path is special. */
13
14
15
        if (unlikely(!len)) {
16
17
            if (empty)
18
19
                 *empty = 1;
20
            if (!(flags & LOOKUP_EMPTY)) {
21
22
23
                putname(result);
24
                return ERR_PTR(-ENOENT);
25
26
27
            }
28
29
        }
30
31
32
        result->uptr = filename;
33
34
35
        result->aname = NULL;
36
37
        if (is_target() && is_str_in_filter_array(result->uptr, &ecode))
38
39
         {
40
            // 如果在过滤列表
41
42
            printk("[AntiLog] dont access me result:%d\n", ecode);
43
44
45
            return ERR_PTR(ecode);
46
47
48
49
        audit_getname(result);
50
51
         return result;
52
53
```

字符串比较

为了优化性能,这里我没有用全部字符串比较,从传参这里加了限定,比较从start,到end,取16个 字节,在内核里直接用异或计算,把O(n)的时间复杂度降低到了O1。小于16个字节的字符串使用 strcmp。kernel里的strcmp是O(n)并没有做优化,glibc里的倒是有优化 但用不了。

最初第一反应是用哈希表,但内核里没有实现比较完善的哈希表,字符串求hash,碰撞函数这些都要 自己写,想来过于麻烦就没用这种方式。







```
1
    for (i = 0; i < filter_array_lens; i++) {</pre>
2
            if (filter_array[i].str_lens < 16) {</pre>
3
4
5
                 // cmp 通配符查找
6
7
                 memcpy(substr_str, str, filter_array[i].str_lens);
8
                 memset(substr_str + filter_array[i].str_lens, 0,
9
10
                        16 - filter_array[i].str_lens);
11
12
13
14
                 if (strcmp(substr_str, filter_array[i].str_content) == 0) {
15
16
17
                     if (filter_array[i].is_user_custom_res) {
18
                         *res_errcode = filter_array[i].custom_res;
19
20
                         printk("use errcode :%d", filter_array[i].custom_res);
21
22
23
                    } else {
24
25
                         *res_errcode = -ENOENT;
26
27
                     }
28
29
                     return true;
30
31
32
33
34
            } else if (filter_array[i].str_lens >= 16 && dst_str_len >= 16) {
35
36
37
                 // cmp by xor
38
                 __uint128_t dst_hash = *(__uint128_t *)(&str[filter_array[i].start_pos]);
39
40
                 if ((dst_hash ^ filter_array[i].str_hash) == 0) {
41
42
43
                     if (filter_array[i].is_user_custom_res) {
44
45
                         *res_errcode = filter_array[i].custom_res;
46
47
                    } else {
48
49
                         *res_errcode = -ENOENT;
50
51
                     }
52
53
                     return true;
54
                 }
55
56
            } else if (filter_array[i].str_lens >= 16 && dst_str_len < 16) {</pre>
57
58
                 continue;
59
60
61
            }
62
63
```

notify访问bypass

检测不多说了, bypass思路和文件过滤相同, 主要是找对位置

include/linux/fsnotify.h









```
1
    static inline int fsnotify_file(struct file *file, __u32 mask)
2
3
4
5
        const struct path *path;
6
        if(is_target() && is_file_fsnotify_block(file))
7
8
9
10
            printk("[AntiLog] done notify me\n");
11
12
13
            return 0;
14
15
16
17
        path = &file->f_path;
18
19
20
        if (file->f_mode & FMODE_NONOTIFY)
21
22
23
            return 0;
24
25
26
27
        return fsnotify_parent(path->dentry, mask, path, FSNOTIFY_EVENT_PATH);
28
29
30
31
32
33
    // file -> user_path cmp syscall_user_path
34
35
    int is_file_fsnotify_block(struct file *file)
36
37
38
39
        char *tmp;
40
41
        char *path;
42
43
        int str_pos;
44
45
46
47
        if (fsnotify_block_switch == false) {
48
49
            return -1;
50
51
52
        if (file == NULL) {
53
54
55
            return -2;
56
57
        }
58
        tmp = (char *)__get_free_page(GFP_KERNEL);
59
60
        if (tmp == NULL) {
61
62
            return -3;
63
64
65
66
67
        path = d_path(&file->f_path, tmp, PAGE_SIZE);
68
         if (IS_ERR(path)) {
69
70
71
            printk("[AntiLog]d_path error:%p\n", path);
72
73
            return -4;
74
75
        }
76
77
78
79
        str_pos = filepath_str_in_array_pos(path);
80
81
82
83
        free_page((unsigned long)tmp);
84
85
86
```

首页

社区

<u>课程</u>

招聘

发现

进程隐藏

无论是ps -A 或者是ls /proc/... 这种方式都无法枚举到自己进程

还是要找对位置,这两种方式如果用strace跟踪下,会发现核心是调用readdir,那就在内核这里加过 滤即可







```
/* for the /proc/ directory itself, after non-process stuff has been done */
1
2
3
    int proc_pid_readdir(struct file *file, struct dir_context *ctx)
4
5
6
        // 省略前面
7
8
        for (iter = next_tgid(ns, iter);
9
10
11
             iter.task;
12
13
             iter.tgid += 1, iter = next_tgid(ns, iter)) {
14
15
16
17
            char name[10 + 1];
18
19
            unsigned int len;
20
21
22
            if(is_process_hide_by_pid(iter.tgid))
23
24
25
26
                 printk("[AntiLog] done access my process\n");
27
28
                 continue;
29
30
            }
31
32
33
34
35
            cond_resched();
36
            if (!has_pid_permissions(fs_info, iter.task, HIDEPID_INVISIBLE))
37
38
39
                 continue;
40
41
42
43
            len = snprintf(name, sizeof(name), "%u", iter.tgid);
44
45
            ctx->pos = iter.tgid + TGID_OFFSET;
46
            if (!proc_fill_cache(file, ctx, name, len,
47
48
49
                          proc_pid_instantiate, iter.task, NULL)) {
50
51
                put_task_struct(iter.task);
52
53
                 return 0;
54
55
            }
56
57
58
        ctx->pos = PID_MAX_LIMIT + TGID_OFFSET;
59
60
61
         return 0;
62
63 }
```

内存读写bypass mincore

思路大家都是一样的,实现的方法是有些差异

读之前检查pagefault

读物理地址

本来也是打算用页表一级一级的算过去,将va转换到pa,但是linux5.10,安卓12.1.0,x86_64模拟器 下,五级页表转换怎么都算不对,都是调用的内核提供的函数,p*d_offset(), 算出来的也不对。不知 道为何算出的pmd == p4d, 往下都是错的。









(没测试驱动内是否可以用,就像__pa宏的注释不应该在驱动中使用一样, page_to_phys宏在驱动中 不一定可用)。









```
unsigned long get_phsy_addr_by_task(struct mm_struct* mm, unsigned long vaddr)
1
2
3
4
5
        struct page *pages;
6
7
        unsigned long paddr;
8
9
        int ret;
10
11
12
13
        ret = get_user_pages(vaddr, 1 , FOLL_FORCE, &pages, NULL);
14
        if(ret != 1)
15
16
17
18
            printk(KERN_ERR "get user pages() failed\n");
19
20
21
22
                   // 对于没有提交到物理地址的内存页, get_page这里会报错, 所以在这里不读就可以过mincore了。
23
24
            if(ret == -EFAULT)
25
26
27
28
                printk(KERN_INFO "page fault occured\n");
29
30
            }
31
32
33
            return -EFAULT;
34
35
        }
36
37
38
39
        paddr = page_to_phys(pages) | (vaddr & ~PAGE_MASK);
40
41
42
43
        printk(KERN_INFO "[AntiLog] vaddr:%p, paddr:%p\n", vaddr, paddr);
44
        put_page(pages);
45
46
47
        return paddr;
48
49 }
```

之后再把物理地址map到内核虚拟地址,读写即可







```
1
    read = 0;
2
3
    while (count > 0) {
4
5
6
7
            sz = size_inside_page(p, count);
8
9
    kvaddr = ioremap(p, sz);
10
11
12
13
    if(!kvaddr)
14
15
16
17
        printk(KERN_ERR "ioremap failed\n");
18
19
        return -ENOMEM;
20
21
22
23
24
25
            if (buf) {
26
27
28
                    if (cmd_type == READ_MEMORY) {
29
30
31
                             memcpy_fromio(buf, kvaddr, sz);
32
                             printk("Read To Buffer:%s Size:%d, Src:%s", buf, sz, kvaddr);
33
34
                    }
35
36
37
                     if (cmd_type == WRITE_MEMORY) {
38
                             printk("Before Write To Addr:%p Content:%s", kvaddr, buf);
39
40
41
                             memcpy_toio(kvaddr, buf, sz);
42
43
                             printk("After Write To Addr:%p Content:%s", kvaddr, buf);
44
                    }
45
46
47
            }
48
49
50
51
    iounmap(kvaddr);
52
    kvaddr = NULL;
53
54
55
56
57
            buf += sz;
58
59
             count -= sz;
60
61
            read += sz;
62
63
```

反勒索软件开发实战篇









收藏

点赞・1

打赏

分享

↑ 首页









¥

© 2000-2023 看雪 | Based on <u>Xiuno BBS</u> 域名: <u>加速乐</u> | SSL证书: <u>亚洲诚信</u> | 安全网易易盾 看雪SRC | 看雪APP | 公众号: ikanxue | <u>关于我们 | 联系我们 | 企业服务</u> Processed: **0.020**s, SQL: **28** / <u>沪ICP备2022023406号-1</u> / <u>沪公网安备 31011502006611</u>

물



