CVE-2013-6282 Analysis Report

Analysis report on Local Privilege Escalation

Caused by Improper Input Validation in the `get_user` and `put_user` API

First Author	Catalin Marinas
Author	iCAROS7 (Homin Rhee)
Data Created	2022.09.20 Tue
Data Version	0.9.9

Index

7. Reference

1. Introduce

CVE-2013-6282는 `arm` 아키텍처의 도메인 전환 기능 사용시 적절치 못한 매개변수 검증으로 인해 권한 상승이 가능한 취약점이다.

이는 2005년 `Linux-2.6.12-rc2` 에서 `arm` 아키텍처용으로 추가된 userland 와 kernel space 간 데이터 전송 메서드인 `get_user()` 및 `put_user()` 내에서 유발 된다. 데이터 전송시 userland와 kernel space의 포인터에 데이터를 읽고 쓰는 과정에서 대상 메모리 주소에 대한 입력 검증이 이루어지지 않아 공격자가 원하는 메모리 주소에 원하는 데이터를 넣을 수 있게 된다.

Linux kernel version 3.5.4 이하의 모든 ARMv6k 및 ARMv7 구성을 사용하는 모든 기기에 해당된다. 이는 실질적으로 2010-13년경 출시된 대부분의 Android 기기에 영향을 미치므로 상당한 위험성을 내제하고 있다. 또한 이미 이를 통해 상당수의 Android 기기가 Local Privilege Escalation (이하 LPE)를 통해 실제로 root 권한을 사용자가 의도적으로 얻어 시스템에 접근하는 rooting 사례가 보고 되었다.

2012년 9월 9일 `Catalin Marinas` 에 의해 최초 보고 되었으며, 동년 동월 10일 Linux main stream에 즉각 커밋 되었다. 이후 다음 해 11월 19일 공개 되며 CVSS 2.0 기준 `7.2` 로 점수를 받았다.

2. Analysis of crash occurrence function

2-1. Basic knowledge

i. Difference of `copy_(to, from)_user()` between `(get, put)_user()`

`copy_(to, from)_user()` 역시 `(get, put)_user()`와 동일하게 userland와 kernel space 간 데이터를 주고 받는데 사용이 가능하다. 차이는 전자의 경우 struct를 포함하여 대다수의 자료형과 구조에 대응이 가능하다. 허나 후자는 char, int 그리고 long 등 간단한 자료형에만 사용이 가능하다.

ARM의 경우 `1`, `2`, `4` byte까지 지원을 하나 2012년 11월 경 추가된 `[PATCH] ARM: add get_user() support for 8 byte types` commit 이후 Linux kernel 3.7 부터 64 bit 자료형인 `8` byte까지 지원한다.

2-2. Code audit

> 하기 모든 Code는 Linux Kernel 3.5.4를 기준으로 한다.

```
1  // File: /arch/arm/include/asm/uaccess.h
2
3  extern int __get_user_1(void *);
4  extern int __get_user_2(void *);
5  extern int __get_user_4(void *);
6
7  #define __get_user_x(__r2,__p,__e,__s,__i...)   \
8       __asm__ __volatile__ (     \
9      __asmeq("%0", "r0") __asmeq("%1", "r2") \
10     "bl __get_user_" #__s   \       // asms, ASM 코드
11     : "=&r" (__e), "=r" (__r2)    \ // output, 결과 출력 변수
12     : "0" (__p)       \             // input, asms에 넘겨줄 입력 변수
13     : __i, "cc")                    // clobber, 상기에 명시되진 않았지만 asms
   로 인해 값이 변하는 변수
```

`__getuser_x()`는 단일 값 전송 메서드이다. Pointer가 정상적으로 할당 되었다면 크기는 자동으로 계산된다.

`__volatile__()`를 통해 인라인 asm 시 최적화 등의 의도치 않은 이동을 방지한다. `__asmeq()`를 통해 두번째 인자 레지스터에 asm 변수에 해당하는 첫번째 인자 asm 변수 값이 정상적으로 mapping 되었나 확인한다. 이때 정상적으로 할당 되지 않았다면 컴파일 작업이 중단된다.

`bl` 명령을 통해 현재의 R15 `PC` 레지스터의 값을 R14 `LR` 레지스터에 복사하여 분기 이후 되돌아올 주소를 현 R15 `PC` 레지스터의 값으로 지정한 뒤 `__get_user_n()` 를 수행한다. 이때 `n` 의 경우 4번째 인자로 받은 `__s` 값을 사용한다. 이때 `equal` 오퍼랜드와 `__p` 로 받은 주소 값을 넘겨준다. 위 asms 연산 중 첫번째 값을 `__e` 주소 값에 이전 값을 버리고 쓰기 전용으로 쓴다. 이후 `__r2` 주소 값에 전자와 동일하게 쓰기 전용으로 쓴다. 모든 asms가 끝나고 난다면 `__i` 주소 값과 `CC` Carry 레지스터의 값이 `0` 으로 변함을 명시한다.

```
#define __put_user_x(__r2,__p,__e,__s)    \
__asm__ __volatile__ (        \
     __asmeq("%0", "r0") __asmeq("%2", "r2")  \
     "bl  __put_user_" #__s   \     // asms, ASM 코드
     : "=&r" (__e)        \         // output, 결과 출력 변수
     : "0" (__p), "r" (__r2)      \ // input, asms에 넘겨줄 입력 변수
     : "ip", "lr", "cc")           // clobber, 상기에 명시되진 않았지만 asms로 인해 값이 변하는 변수
```

`__get_user_x()` 와 비슷한 메커니즘으로 동작하지만 차이점만 짚어보겠다. asms 수행시 `equal` 오퍼랜드와 `__p` 로 받은 주소 값을 첫번째로, 쓰기 전용으로 이전 값을 버리고 `__r2` 주소 값을 넘겨준다. 이후 연산 중 `__e` 의 이전 값을 버리고 새로운 값을 쓴다. 모든 asms가 끝나고 난다면 R12 `IP` Intra scratch 레지스터와 R14 `LR` Link Register의 복귀 주소가 바뀜을 명시한다.

```
#define get_user(x,p)           \
  ({              \
    register const typeof(*(p)) __user *__p asm("r0") = (p);\
    register unsigned long __r2 asm("r2");      \
    register int __e asm("r0");       \
    switch (sizeof(*(__p))) {       \
    case 1:             \
      __get_user_x(__r2, __p, __e, 1, "lr");    \
           break;           \
    case 2:             \
      __get_user_x(__r2, __p, __e, 2, "r3", "lr");  \
      break;            \
    case 4:             \
           __get_user_x(__r2, __p, __e, 4, "lr");    \
```

```
15        break;            \
16      default: __e = __get_user_bad(); break;     \
17      }             \
18      x = (typeof(*(p))) __r2;        \
19      __e;              \
20    })
```

실질적으로 사용되는 `get_user()` 이다. userland 상의 포인터로부터 단일 값 `x` 를 가져온다. 전반적인 흐름은 사용될 변수들이 선언되고 크기에 따라 switch 문을 통해 `get_user_n()` 으로 분배된다.

`register` 키워드로 R0 레지스터에 static 하게 저장이 되는 변수 `__p` 를 하나 만들어준다. 형식은 매개변수로 받은 `p` 와 같은 형식으로 한다. 동일하게 `R2` 레지스터에 저장되는 unsigned long 형식의 변수 `__r2` 를 선언한다. R0 레지스터에 저장되는 `__e` 도 하나 선언 한다.

`sizeof()` 를 사용하여 매개변수로 받은 `p` 의 크기에 따라 switch-case 문을 실행한다. 이때 올바르지 않은 형식의 경우 `__get_user_bad()` 를 호출하여 원치 않는 메모리 읽기 쓰기를 차단한다.

이후 새롭게 userland로부터 가져온 R2 레지스터의 값을 `typeof()` 로 자료형을 맞추어 x에 대입한다.

```
1  #define put_user(x,p) \
2    ({  \
3      register const typeof(*(p)) __r2 asm("r2") = (x); \
4      register const typeof(*(p)) __user *__p asm("r0") = (p);\
5      register int __e asm("r0");       \
6      switch (sizeof(*(__p))) {       \
7      case 1:             \
8        __put_user_x(__r2, __p, __e, 1);    \
9        break;            \
10     case 2:             \
11       __put_user_x(__r2, __p, __e, 2);    \
12       break;            \
13     case 4:             \
14       __put_user_x(__r2, __p, __e, 4);    \
15       break;            \
16     case 8:             \
```

```
17        __put_user_x(__r2, __p, __e, 8);    \
18        break;            \
19      default: __e = __put_user_bad(); break;     \
20      }             \
21      __e;              \
22    })
```

`put_user()` 메서드도 `get_user()` 메서드와 동일한 메커니즘으로 동작한다. 차이 점만 짚어보자면 R2 레지스터에 static 값으로 userland로 넘겨줄 포인터 `p` 의 자료형에 따라 `__r2` 가 선언 되고 해당 값에 매개변수로 들어온 `x` 의 주소를 대입한다. 이후 동일하게 R0 레지스터에 static 한 `__p` 포인터를 선언하여 `p` 의 주소를 대입한다.

```
1  // File: /arch/arm/lib/getuser.S
2
3  #include <linux/linkage.h>
4  #include <asm/errno.h>
5  #include <asm/domain.h>
6
7  ENTRY(__get_user_1)
8  1: TUSER(ldrb)  r2, [r0]
9    mov r0, #0
10   mov pc, lr
11 ENDPROC(__get_user_1)
12
13 ENTRY(__get_user_2)
14 #ifdef CONFIG_THUMB2_KERNEL
15 2: TUSER(ldrb)  r2, [r0]
16 3: TUSER(ldrb)  r3, [r0, #1]
17 #else
18 2: TUSER(ldrb)  r2, [r0], #1
19 3: TUSER(ldrb)  r3, [r0]
20 #endif
21 #ifndef __ARMEB__
22   orr r2, r2, r3, lsl #8
23 #else
```

```
24    orr r2, r3, r2, lsl #8
25  #endif
26    mov r0, #0
27    mov pc, lr
28  ENDPROC(__get_user_2)
29
30  ENTRY(__get_user_4)
31  4: TUSER(ldr) r2, [r0]
32    mov r0, #0
33    mov pc, lr
34  ENDPROC(__get_user_4)
```

`getuser.S` 에서는 ENTRY 매크로를 통해 각 크기 name label을 보여줄 수 있게 정의를 해두었다.

```
1   // File: /arch/arm/lib/putuser.S
2
3   #include <linux/linkage.h>
4   #include <asm/errno.h>
5   #include <asm/domain.h>
6
7   ENTRY(__put_user_1)
8   1: TUSER(strb)  r2, [r0]
9     mov r0, #0
10    mov pc, lr
11  ENDPROC(__put_user_1)
12
13  ENTRY(__put_user_2)
14    mov ip, r2, lsr #8
15  #ifdef CONFIG_THUMB2_KERNEL
16  #ifndef __ARMEB__
17  2: TUSER(strb)  r2, [r0]
18  3: TUSER(strb)  ip, [r0, #1]
19  #else
20  2: TUSER(strb)  ip, [r0]
21  3: TUSER(strb)  r2, [r0, #1]
22  #endif
```

```
23  #else /* !CONFIG_THUMB2_KERNEL */
24  #ifndef __ARMEB__
25  2: TUSER(strb)  r2, [r0], #1
26  3: TUSER(strb)  ip, [r0]
27  #else
28  2: TUSER(strb)  ip, [r0], #1
29  3: TUSER(strb)  r2, [r0]
30  #endif
31  #endif  /* CONFIG_THUMB2_KERNEL */
32    mov r0, #0
33    mov pc, lr
34  ENDPROC(__put_user_2)
35
36  ENTRY(__put_user_4)
37  4: TUSER(str) r2, [r0]
38    mov r0, #0
39    mov pc, lr
40  ENDPROC(__put_user_4)
41
42  ENTRY(__put_user_8)
43  #ifdef CONFIG_THUMB2_KERNEL
44  5: TUSER(str) r2, [r0]
45  6: TUSER(str) r3, [r0, #4]
46  #else
47  5: TUSER(str) r2, [r0], #4
48  6: TUSER(str) r3, [r0]
49  #endif
50    mov r0, #0
51    mov pc, lr
52  ENDPROC(__put_user_8)
```

2-3. Vulnerability analysis

실질적으로 취약한 code는 하기와 같다.

4. Proof of concept

5. Patch for the vulnerability

6. Conclusion

7. Reference

- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-6282
- https://cve.report/CVE-2013-6282
- https://nvd.nist.gov/vuln/detail/CVE-2013-6282
- https://ubuntu.com/security/CVE-2013-6282
- https://access.redhat.com/security/cve/cve-2013-6282
- https://security-tracker.debian.org/tracker/CVE-2013-6282
- https://vuldb.com/ko/?id.11226
- https://www.mend.io/vulnerability-database/CVE-2013-6282
- https://mirrors.edge.kernel.org/pub/linux/kernel/v3.x/ChangeLog-3.5.5

- https://github.com/torvalds/linux/commit/8404663f81d212918ff85f493649a7991209fa04
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8404663f81d212918ff85f493649a7991209fa04
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/arch/arm/include/asm/uaccess.h?id=8404663f81d212918ff85f493649a7991209fa04

- https://web.archive.org/web/20140327052415/https://www.codeaurora.org/projects/security-advisories/missing-access-checks-putusergetuser-kernel-api-cve-2013-6282
- https://lore.kernel.org
- https://developer.arm.com/documentation/ddi0406/c/System-Level-Architecture/System-Control-Registers-in-a-VMSA-implementation/VMSA-System-control-registers-descriptions--in-register-order/DACR--Domain-Access-Control-Register--VMSA
- https://developer.arm.com/documentation/ddi0388/i/system-control/register-summary/virtual-memory-control-registers
- https://wiki.kldp.org/KoreanDoc/html/EmbeddedKernel-KLDP
- https://blog.csdn.net/ce123_zhouwei/article/details/8209702

- https://elixir.bootlin.com
- https://codebrowser.dev/