CVE-2021-21781

Analysis Report

Analysis report on ARM SIGPAGE data disclosure

Caused by use of uninitialized page

In get_signal_page(void) of Kernel Signal Management

First reporter: `Lilith >_>` of "Cisco Talos Intelligence Group"

Author: iCAROS7

Data Created: 2022.06.21 Tue

Data Version: 1.1.1

Index

1. Introduce

CVE-2021-21781 은 userland 상의 application 을 Return-Oriented Programing (이하 ROP)을 통해 information leak 이 가능한 취약점이다.

이는 process 가 최초 signal initialization 시 ARM command & signal handler data 를 주고받기 위한 page 를 random 한 offset 을 포함해 allocating 하는 과정에서 trigger 된다. Uninitialized 된 address 가 포함된 page 를 allocating 받아 해당 page 에 존재하는 기존의 타 SIGPAGE Data 에 access 가능하게 되며 이루어진다.

Linux kernel version 4.0 ~ 4.14.221, ~ 4.19.176, 5.0 ~ 5.4.98, ~ 5.10.16 그리고 ~ 5.11-rc4 까지의 범위를 지닌다.

2021 년 1 월 28 일 `Cisco Talos` 팀의 `Lilith >_>` 에 의해 발견된 후 각 vendor 에게 first report 되었으며 2022 년 2 월 5 일 Linux main branch 에 patch 가 commit & merge 되었다. 이후 동년 6 월 25 일 Public Release 되며 CVSS 3.0 기준 `3.3 Low`로 scoring 되었다.

2. Analysis of crash occurrence function

i. Basic knowledge

Linux kernel 에서의 signal 은 process - process 혹은 kernel - process 간 주고 받는 정보 이다.

 # /Linux/arch/arm/kernel/process.c

```
01  static const struct vm_special_mapping sigpage_mapping = {
02          .name = "[sigpage]",
03          .pages = &signal_page,                          // [1]
04          .mremap = sigpage_mremap,
05  };
06
07
08  static struct page *signal_page;
09  extern struct page *get_signal_page(void);
10  int arch_setup_additional_pages(struct linux_binprm *bprm,
            int uses_interp)
11  {
12          struct mm_struct *mm = current->mm;
13          struct vm_area_struct *vma;
14          unsigned long npages;
15          unsigned long addr;
16          unsigned long hint;
17          int ret = 0;
18
19          if (!signal_page)
20                  signal_page = get_signal_page();    // [2]
21          if (!signal_page)
22                  return -ENOMEM;
23
24          npages = 1; /* for sigpage */
25          npages += vdso_total_pages;
26
27          if (down_write_killable(&mm->mmap_sem))
28                  return -EINTR;
29          hint = sigpage_addr(mm, npages);
30          addr = get_unmapped_area(NULL, hint,
                    npages << PAGE_SHIFT, 0, 0);
31       if (IS_ERR_VALUE(addr)) {
32                  ret = addr;
33                  goto up_fail;
34          }
35
36          vma = _install_special_mapping(mm
                    , addr, PAGE_SIZE, VM_READ |
                    VM_EXEC | VM_MAYREAD | VM_MAYWRITE |
                    VM_MAYEXEC, &sigpage_mapping);       // [3]
```

 # /Linux/mm/mmap.c

```
01          struct vm_area_struct *_install_special_mapping(
02          struct mm_struct *mm,
03          unsigned long addr, unsigned long len,
04          unsigned long vm_flags,
                const struct vm_special_mapping *spec)       // [4]
05          {
06          return __install_special_mapping(mm,
                addr, len, vm_flags, (void *)spec,
                &special_mapping_vmops);
07          }
```

이 중 후자에서는 `process.c` 상의 Line 20, [2]와 같이 최초 initialization 시 데이터를 주고 받기 위한 일정 PAGE 를 프로세스에게 allocating 한다. 이 과정에서 get_signal_page() function 을 통해 page structor 를 Return 받는다.

이를 기반으로 Line 36, [3]에서 `sigpage_mapping` struct 가 `mm/mmap.c` 상 `_install_special_mapping()`의 arguments 로 이용되는 과정 중 Line 3, [1]와 같이 `signal_page`가 `.pages`에 reference 된다.

이후 `mmap.c` 상 Line 04, [4]에서 `__install_special_mapping()` arguments 로 전달되어 memory 의 available address 에 allocate 된다.

ii. Code audit

 # /Linux/arch/arm/kernel/signal.c

```
01  struct page *get_signal_page(void)
02  {
03          unsigned long ptr;
04          unsigned offset;
05          struct page *page;
06          void *addr;
07
08          page = alloc_pages(GFP_KERNEL, 0);
09
10          if (!page)
11                  return NULL;
12
13          addr = page_address(page);
14
15          /* Give the signal return code some randomness */
16          offset = 0x200 + (get_random_int() & 0x7fc);
17          signal_return_offset = offset;
18
19          /* Copy signal return handlers into the page */
20                  memcpy(addr + offset,
                    sigreturn_codes, sizeof(sigreturn_codes));
21
22          /* Flush out all instructions in this page */
23          ptr = (unsigned long)addr + offset;          // [1]
24          flush_icache_range(ptr,
                    ptr + sizeof(sigreturn_codes));      // [2]
25
26          return page;
27  }
```

Line. 7 에서 alloc_pages() function 을 통해 logical page 를 return 받는다. 이후 Line. 13 과 같이 page_address() function 을 통해 첫번째 page 의 logical address 를 return 받아 `addr` pointer 에 저장한다.

Line. 16 에서 기존에 받은 `SIGRET_CODE` 의 내용을 복사하기 위한 새로운 영역을 만들기 위해 특정 offset 을 계산하여 unsigned int 형으로 `offset` 및 `signal_return_offset`에 저장 한다. 이는 int 형이 4byte 씩 메모리를 사용하고, Linux kernel 에서 ARM architecture 의 unit of page allocating 인 4KB 로 잘라 생성하기 위함이다.

Line. 20 에서 `addr` 로부터 4byte * `offset` 의 합 address 에 `sigreturn_codes`의 명령어들을 크기만큼 복사한다.

Line. 23 에서는 Line. 20 에서 복사된 address 를 `ptr` 변수에 저장한다.

Line. 24 에서 flush_icache_range() function 을 통해 `ptr` ~ (`ptr` + sizeof(`sigreturn_codes`)) 까지의 Kernel 이 명령어 처리를 위한 Instruction Cache 임을 명시함과 동시에 해당 영역을 flushing 한다.

iii. Vulnerability analysis

```
ptr = (unsigned long)addr + offset;
flush_icache_range(ptr, ptr + sizeof(sigreturn_codes));
```

실질적으로 vulnerability 인 code 의 경우 위와 같다. `sigreturn_codes` 는 unsigned long 형식으로 extern type 으로 declare 이 되어있다. sizeof() function 는 pointer 의 경우 해당 pointer 의 타입의 size 가 반환된다.

허나 만일 sizeof(`sigreturn_codes`)가 0 인 경우에는 정상적으로 flush 가 수행되지 않으며 abnormal 한 `page` object 가 `process.c` 의 arch_setup_additional_pages()로 return 되며 available memory address 로 mapping 되는 과정에서 flush 가 되지 않아 uninitialized 된 메모리가 allocating 될 수 있다.

단, 이는 상기한 allocating 과정 상에서 virtual memory table 상에 available 하다 판단되는 address + offset 에 이전의 flushing 되지 않은 data 가 남아 있어야만 read 가 가능하다.

Flushing 되지 않은 data 가 남아 있는 case 의 경우 제 3 자가 ROP attack 을 통해 특정 offset 상의 kernel memory data 를 read 가능한 상황이 된다.

3. Crash inducement

i. Host PC Specification

Host OS Version	: Ubuntu 20.04.04 LTS (Focal Fossa)
Host Kernel Version	: Linux 5.13.0-25-generic aarch64
Host CPU	: Apple M1 @3.2Ghz
Host RAM	: LPDDR4X 4266Mhz 16GB
Swap File Size	: 8,192MB

ii. Target Specification

Tested Architecture	: arm (ARMv7, Cotrex-a15)
Tested Virtual Env	: vexpress-a15
Tested Kernel Ver	: Linux 5.4.66-gerneric
Tested Qemu Ver	: 4.2.1
Enabled Sanitizer	: KGDB (Kernel Gnu Debuger)

iii. Exploit

Linux 상에서 현재 실행 중인 process 의 정보는 `/proc/<pid>` 에서 확인이 가능하다.

```
# ps -e | grep sh
  164 root     sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
  167 root     -sh
  202 root     grep sh

# cat /proc/167/status | grep Name
Name:  sh
```

이중 `maps` 파일을 확인 시 현재 process 에 대한 memory map information 을 얻을 수 있다.

```
# cat /proc/167/maps
0046d000-00533000 r-xp 00000000 b3:00 17       /bin/busybox
00542000-00544000 r--p 000c5000 b3:00 17       /bin/busybox
00544000-00545000 rw-p 000c7000 b3:00 17       /bin/busybox
00545000-0054a000 rw-p 00000000 00:00 0        [heap]
76e9b000-76f03000 r-xp 00000000 b3:00 173      /lib/libuClibc-1.0.41.so
76f03000-76f12000 ---p 00000000 00:00 0
76f12000-76f13000 r--p 00067000 b3:00 173      /lib/libuClibc-1.0.41.so
76f13000-76f14000 rw-p 00068000 b3:00 173      /lib/libuClibc-1.0.41.so
76f14000-76f2a000 rw-p 00000000 00:00 0
76f2a000-76f30000 r-xp 00000000 b3:00 163      /lib/ld-uClibc-1.0.41.so
76f3e000-76f40000 rw-p 00000000 00:00 0
76f40000-76f41000 r--p 00006000 b3:00 163      /lib/ld-uClibc-1.0.41.so
76f41000-76f42000 rw-p 00007000 b3:00 163      /lib/ld-uClibc-1.0.41.so
7ec5e000-7ec7f000 rw-p 00000000 00:00 0        [stack]
7ed2c000-7ed2d000 r-xp 00000000 00:00 0        [sigpage]
7ed2d000-7ed2e000 r--p 00000000 00:00 0        [vvar]
7ed2e000-7ed2f000 r-xp 00000000 00:00 0        [vdso]
ffff0000-ffff1000 r-xp 00000000 00:00 0        [vectors]
```

예시로 pid 167 로 실행 중인 `/bin/sh`에 대한 memory map information 을 출력 시 위와 같다. 여기서 `[sigpage]` data 가 담긴 0x7ed2c000 - 0x7ed2d000 의 virtual memory 가 allocating 된 것을 확인 가능 하다.

KGDB 를 통해 해당 memory address 의 data 를 확인 시 위와 같다.

4. Patch for the vulnerability

```
#       /Linux/arch/arm/kernel/signal.c

        struct page *get_signal_page(void)
        {
        unsigned long ptr;
        unsigned offset;
        struct page *page;
        void *addr;

        addr = page_address(page);

        // memset32 를 통하여 uint32_t 로 Memory area Filling
+       memset32(addr, __opcode_to_mem_arm(0xe7fddef1),
+                PAGE_SIZE / sizeof(u32));              // [1]
+
        /* Give the signal return code some randomness */
        offset = 0x200 + (get_random_int() & 0x7fc);
        signal_return_offset = offset;

        memcpy(addr + offset, sigreturn_codes, sizeof(sigreturn_codes));

        // offset 값을 통한 ROP Attack Prevent
+       ptr = (unsigned long)addr;                     // [2]
        // PAGE_SIZE 단위로 Flushing
+       flush_icache_range(ptr, ptr + PAGE_SIZE);      // [3]

        return page;
 }
```

[1]과 같이 사전에 `memset32()` function 을 통해 uint32_t 로 memory aera 를 4 byte 단위로 filling 한다.

[2]에서 `offset`를 지우는 것으로 인해 trigger 될 수 있는 ROP attack 을 prevent 가능하다. 이후 [3]에서 기존 `sizeof`가 아닌 `PAGE_SIZE` 단위로 flushing 을 진행하여 uninitialization 을 prevent 한다.

5. Conclusion

위 exploit 은 실질적으로 exploitability 한 상황이 local 상에서 쉬운 난이도로 별도의 privileges 이나 user interaction 없이 가능하지만 system 의 confidentiality / integrity / availability 에 영향을 주지 않아 낮게 scoring 되었다. USD $0 ~ $5K 로 추정 bounty 만 존재할 뿐 정확한 bounty 정보는 공개되지 않았다.

6. Reference

- https://github.com/torvalds/linux/commits/master/arch/arm/kernel/signal.c
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0185
- https://ubuntu.com/security/CVE-2021-21781
- https://elixir.bootlin.com/linux
- https://www.cve.org/CVERecord?id=CVE-2021-21781
- https://nvd.nist.gov/vuln/detail/CVE-2021-21781
- https://cve.report/CVE-2021-21781
- https://access.redhat.com/security/cve/cve-2021-21781
- https://bugzilla.redhat.com/show_bug.cgi?id=1981950
- https://vulners.com/cve/CVE-2021-21781
- https://books.google.co.kr/books/about/코드로_알아보는_ARM_리눅스_커.html
- https://cpuu.postype.com/post/9075747
- https://blog.daum.net/tlos6733/188
- https://wogh8732.tistory.com/395
- https://github.com/google/syzkaller/blob/master/docs/linux/setup_linux-host_qemu-vm_arm-kernel.mds