# 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:** 1.2.0

# 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
            // logical page return
08          page = alloc_pages(GFP_KERNEL, 0);          // [1]
09
10          if (!page)
11                  return NULL;
12
            // Save address to *addr
13          addr = page_address(page);                  // [2]
14
            // Set random offset for memcopy()
15          /* Give the signal return code some randomness */
16          offset = 0x200 + (get_random_int()
                    & 0x7fc);                           // [3]
17          signal_return_offset = offset;
18
            // Copy of sigreturn codes
19          /* Copy signal return handlers into the page */
20          memcpy(addr + offset, sigreturn_codes,
                    sizeof(sigreturn_codes));           // [4]
21
            // Flushing ptr ~ ptr + sigreturn codes
22          /* Flush out all instructions in this page */
23          ptr = (unsigned long)addr + offset;         // [5]
24          flush_icache_range(ptr,
                    ptr + sizeof(sigreturn_codes));     // [6]
25
26          return page;
27  }
```

Line.8 , [1]에서 alloc_pages() function 을 통해 `GFP_KERNEL` type 으로 logical page 를 return 받는다.

이후 Line. 13, [2] 과 같이 page_address() function 을 통해 첫번째 page 의 logical address 를 return 받아 `addr` pointer 에 저장한다.

Line. 16, [3] 에서 기존에 받은 `SIGRET_CODE` 의 내용을 복사하기 위한 새로운 aera 을 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 split 하여 생성하기 위함이다.

Line. 20, [4]에서 `addr` 로부터 4byte * `offset` 의 합 address 에 `sigreturn_codes`의 명령어들을 `sigreturn_codes`의 size 만큼 copy 한다.

Line. 23, [5]에서는 [4]에서 복사된 address 를 `ptr` 변수에 unsigned long type 으로 conversion 하여 저장한다.

Line. 24, [6]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

실질적으로 vulnerability 인 code 의 경우 하기와 같다.

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

`sigreturn_codes` 는 unsigned long 형식으로 extern type 으로 declare 가 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.

실제 해당 area 의 data 를 확인하기 위해서는 Kernel Gnu Debuger (이하 KGDB)를 통해 확인이 가능하다.

이를 위해 vulnerability kernel version 인 5.4.66-gerneric 을 kgdb 관련 config 를 enable 한 상태로 build 한다. 이후 buildroot 2022.02.3 image 와 함께 구성 함을 통해 environments 구축이 가능하다.

```
 -append "kgdboc=ttyS0, 115200 kgdbwait"

 [      0.000000] Kernel command line: earlyprintk=serial console=ttyAMA0
root=/dev/mmcblk0 kgdboc=ttyS0, 115200 kgdbwait
```

위와 같이 boot arguments 에 `kgdboc`와 `kgdbwait` 인자 setup 을 통해 kgdb parameter 에 통신에 사용할 teletypewriter 를 boot-up 중 로 setup 할 수 있다. 이후 바로 connection 을 establishment 할 수 있다. 본 환경에서는 `ttyS0` serial communication 을 115200 baud 로 진행한다.

```
[    1.005706] 8<--- cut here ---
[    1.005768] Unable to handle kernel paging request
                at virtual address fee00001
[    1.005918] pgd = (ptrval)
[    1.005969] [fee00001] *pgd=00000000
[    1.006222] Internal error: Oops: 5 [#1] SMP ARM
[    1.006432] KGDB: re-enter exception: ALL breakpoints killed
[    1.006831] ---[ end trace d73e3b3eb7b3fa24 ]---
[    1.006994] note: swapper/0[1] exited with preempt_count 3
[    1.007093] BUG: sleeping function called
                from invalid context at include/linux/percpu-rwsem.h:38
[    1.007192] in_atomic(): 0, irqs_disabled(): 128
                , non_block: 0, pid: 1, name: swapper/0
[    1.007392] CPU: 1 PID: 1 Comm: swapper/0
                Tainted: G     D         5.4.66 #2
[    1.007478] Hardware name: ARM-Versatile Express
[    1.008148] [<c021c410>] (unwind_backtrace) from
                [<c0215e48>] (show_stack+0x10/0x14)
[    1.008394] [<c0215e48>] (show_stack) from
                [<c0e288ac>] (dump_stack+0xbc/0xd0)
[    1.008491] [<c0e288ac>] (dump_stack) from
                [<c027c2b0>] (___might_sleep+0x128/0x170)
[    1.008594] [<c027c2b0>] (___might_sleep) from
                [<c0263c30>] (exit_signals+0x34/0x2c0)
[    1.008941] [<c0263c30>] (exit_signals) from
                [<c0255c54>] (do_exit+0xdc/0xb00)
[    1.009112] [<c0255c54>] (do_exit) from
                [<c02207d0>] (do_page_fault+0x0/0x3a0)
[    1.009334] [<c02207d0>] (do_page_fault) from [<ee8c9b34>] (0xee8c9b34)
[    1.009543] Kernel panic - not syncing: Attempted
                to kill init! exitcode=0x00000009
[    1.009712] 8<--- cut here ---
[    1.009771] Unable to handle kernel paging request
                at virtual address fee00001
[    1.009846] pgd = (ptrval)
[    1.009885] [fee00001] *pgd=00000000
[    1.009956] Internal error: Oops: 5 [#2] SMP ARM
[    1.010055] CPU: 1 PID: 1 Comm: swapper/0 Tainted:
                G     D W       5.4.66 #2
[    1.010131] Hardware name: ARM-Versatile Express
[    1.010198] [<c021c410>] (unwind_backtrace) from
                [<c0215e48>] (show_stack+0x10/0x14)
[    1.010278] [<c0215e48>] (show_stack) from
                [<c0e288ac>] (dump_stack+0xbc/0xd0)
```

```
[    1.010356] [<c0e288ac>] (dump_stack) from
                [<c02ffef8>] (kgdb_handle_exception+0x1e8/0x238)
[    1.010450] [<c02ffef8>] (kgdb_handle_exception) from
                [<c021b9ac>] (kgdb_notify+0x24/0x38)                // [1]
[    1.010531] [<c021b9ac>] (kgdb_notify) from
                [<c0276c68>] (notifier_call_chain+0x48/0x84)
[    1.010620] [<c0276c68>] (notifier_call_chain) from
                [<c0276cd8>] (__atomic_notifier_call_chain+0x34/0x50)
[    1.010710] [<c0276cd8>] (__atomic_notifier_call_chain) from
                [<c0277418>] (notify_die+0x60/0x88)
[    1.011003] [<c0277418>] (notify_die) from [<c0215f74>] (die+0x128/0x374)
[    1.011068] [<c0215f74>] (die) from
                [<c02207c0>] (__do_kernel_fault.part.0+0x78/0x88)   // [2]
[    1.011156] [<c02207c0>] (__do_kernel_fault.part.0) from
                [<c0220bd0>] (do_sect_fault+0x0/0x10)
[    1.011298] [<c0220bd0>] (do_sect_fault) from [<c0207fb8>] (0xc0207fb8)
[    1.011381] Kernel panic - not syncing: Recursive entry to debugger
[    2.300654] SMP: failed to stop secondary CPUs
[    2.300907] ---[ end Kernel panic - not syncing:
                Recursive entry to debugger ]---                    // [3]
```

실제 environments 구성 할 경우 상기와 같이 [1] `kgdb_handle_exception`이 일어난다. 이로 인해 [2]와 같이 kernel 이 실제 존재하지 않는 page 에 access 를 시도하며 `__do_kernel_fault()`가 호출 된다.

결국 [3]에서 확인 가능 한 것 처럼 `Recursive entry to debugger` 로 인해 trigging 된다.

# 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

상기 Exploit 상의 문제를 해결해보기 위해 boot arguments 에서 `kgdboc` 관련 flag 를 제거하고 `kgdbwait` 상태만 enable 시켜 boot-up 한다.

```
echo 'ttyS0' > /sys/module/kgdboc/parameters/kgdboc
```

위와 같이 `kgdboc` parameter 에 communication 에 사용할 teletypewriter 를 지정할 경우 system 이 성공적으로 interrupt 되지만 이내 다시 정상적으로 boot-up 된다.

동일한 환경 구성으로 x86-64 architecture 와 first reporter 의 environment 상에서는 issue 가 없는 것으로 확인 된다. 이는 author local 상의 issue 로 판단 된다.

또한 본 vulnerability code 의 경우 System 에 직접적인 영향보다는 ROP attack 을 통해 극히 일부의 case 에서 possibility 를 보여준다. 이로 인해 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 존재 할 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