

一.前言

由于Android系统与Linux系统具有较强的联系，相应的实现也十分类似，故首先实现Linux下的Hook技术用做前期练习。在Linux系统中利用Ptrace()函数可以很轻松的对相应的进程进行修改和调试。本次实现的大致流程如图。



二.Ptrace()函数介绍

1.函数原型:

```
long ptrace(enum __ptrace_request request,pid_t pid, void *addr,void *data); ``
```

ptrace 提供了一种父进程可以控制子进程运行，并可以检查和改变它的核心image。它主要用于实现断点调试。一个被跟踪的进程运行中，直到发生一个信号。则进程被中止，并且通知其父进程。在进程中止的状态下，进程的内存空间可以被读写。父进程还可以使子进程继续执行，并选择是否是否忽略引起中止的

信号。

(1). 第一个参数决定了 **ptrace** 的行为，其值有：

a). **ptrace_attach**: 跟踪指定 *pid* 进程。*pid* 表示被跟踪进程。被跟踪进程将成为当前进程的子进程，并进入中止状态

b). **ptrace_getregs**: 读取寄存器值，*pid* 表示被跟踪的子进程，*data* 为用户变量地址用于返回读到的数据。此功能将读取所有 17 个基本寄存器的值。

c). **ptrace_setregs**: 设置寄存器值，*pid* 表示被跟踪的子进程，*data* 为用户数据地址。此功能将设置所有 17 个基本寄存器的值。

d). **ptrace_detach**: 结束跟踪。*pid* 表示被跟踪的子进程。结束跟踪后被跟踪进程将继续执行。

e). **ptrace_cont**: 继续执行。*pid* 表示被跟踪的子进程，*signal* 为 0 则忽略引起调试进程中止的信号，若不为 0 则继续处理信号 *signal*。

以上只列出了本次测试所用到的请求，更多请查看以下链接：

http://blog.sina.com.cn/s/blog_4ac74e9a0100n7w1.html

(2). 第二个参数决定了 **ptrace** 所要操作的进程，第三个参数指定了操作数据的地址，第四个参数指定了数据的长度

三.应用 **ptrace** 函数

利用 **ptrace** 函数将目标函数暂停。

1. 目标函数进程 **des.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<string.h>
#include<sys/types.h>
#include<unistd.h>
const int size=sizeof(char);
int main()
{
    int i;
    for(i = 0;i < 10;i++) {
        printf("My counter: %d \n", i);
        sleep(2);
    }
    return 0;
}
```

2. **ptrace** 利用函数 **ptrace.c**

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include<sys/user.h>
#include<sys/reg.h>
#include <sys/wait.h>
#include <unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
const int long_size = sizeof(long);
//这个函数用来获取rip之后三个指令的值
void getdata(pid_t child, long addr,char *str, int len)
{
    char *laddr;
    int i, j;
    union u {
        long val;
        char chars[long_size];
    }data;

    i = 0;
    j = len / long_size;
    laddr = str;

    while(i < j) {
```

```

        data.val = ptrace(PTRACE_PEEKDATA, child,
                        addr + i * 8, NULL);
        memcpy(laddr, data.chars, long_size);
        ++i;
        laddr += long_size;
    }
    j = len % long_size;
    if(j != 0) {
        data.val = ptrace(PTRACE_PEEKDATA, child,
                        addr + i * 8, NULL);
        memcpy(laddr, data.chars, j);
    }
    str[len] = '\0';
    printf("get    %c,%c,%c \n",laddr[0],laddr[1],laddr[2]);    //this
}

//以下函数用来设置rip之后三个指定的值
void putdata(pid_t child, long addr,
            char *str, int len)
{
    char *laddr;
    int i, j;
    union u {
        long val;
        char chars[long_size];
    }data;

    i = 0;
    j = len / long_size;
    laddr = str;
    while(i < j) {
        memcpy(data.chars, laddr, long_size);
        ptrace(PTRACE_POKEDATA, child,
                addr + i * 8, data.val);
        ++i;
        laddr += long_size;
    }
    j = len % long_size;
    if(j != 0) {
        memcpy(data.chars, laddr, j);
        ptrace(PTRACE_POKEDATA, child,
                addr + i * 8, data.val);
    }
    printf("put    %c,%c,%c \n",laddr[0],laddr[1],laddr[2]);//this
}

int main(int argc, char *argv[])
{
    pid_t traced_process;
    struct user_regs_struct regs;
    long ins;

    /* int 0x80, int3 */

    char code[] = {0xcd,0x80,0xcc,0};    //这个是实现int3中断
    char backup[4];
    if(argc != 2) {
        printf("Usage: %s <pid to be traced> ", argv[0]);
        exit(1);
    }
    traced_process = atoi(argv[1]);
    printf("pid    %d \n",traced_process);
    if((ptrace(PTRACE_ATTACH, traced_process,NULL, NULL))!=0)
        printf("attch failed");

    wait(NULL);
    if((ptrace(PTRACE_GETREGS, traced_process, NULL, &regs))!=0)
        printf("get failed");

    /* Copy instructions into a backup variable */

    getdata(traced_process, regs.rip, backup, 3);

    /* Put the breakpoint */

    putdata(traced_process, regs.rip, code, 3);

    /* Let the process continue and execute the int 3 instruction */

    if((ptrace(PTRACE_CONT, traced_process, NULL, NULL))!=0)
        printf("continue failed");
    wait(NULL);
    printf("The process stopped, putting back "
           "the original instructions ");
    printf("Press <enter> to continue ");
    getchar();
    putdata(traced_process, regs.rip, backup, 3);

    /* Setting the rip back to the original instruction to let the process continue*/

    if((ptrace(PTRACE_SETREGS, traced_process, NULL, &regs))!=0)
        printf("set reg failed");
    else
        printf("set succeed \n");

    if((ptrace(PTRACE_DETACH, traced_process,NULL, NULL))!=0)
        printf("detach failed");
    else
        printf("detach succeed \n");
}

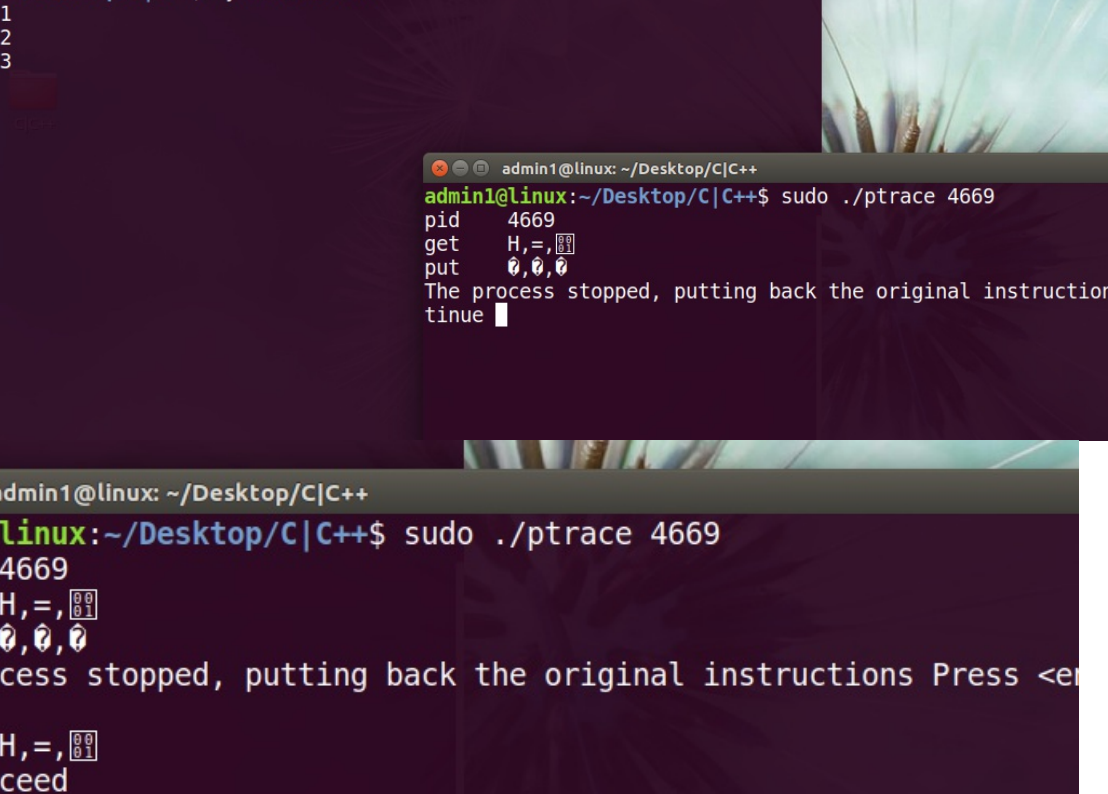
```

3.测试

```
gcc -o des des.c gcc -o ptrace ptrace.c
```

在终端中输入 `./des &` 使得 `des` 在后台运行并且返回相应的 `pid`，再在终端中输入 `sudo ./ptrace pid(des的pid)` 即可发现 `ptrace` 函数成功修改了 `des` 函数的代码逻辑。在未运行 `ptrace` 函数时 `des` 函数会顺利运行直至结束，但是运行 `ptrace` 函数后 `des` 函数会发生中断。至此结束 `ptrace` 函数的学习。

...



```
admin1@linux: ~/Desktop/C|C++
admin1@linux:~/Desktop/C|C++$ ./des &
[1] 4669
admin1@linux:~/Desktop/C|C++$ My counter: 0
My counter: 1
My counter: 2
My counter: 3

```

```
admin1@linux: ~/Desktop/C|C++
admin1@linux:~/Desktop/C|C++$ sudo ./ptrace 4669
pid    4669
get     H,=,00
put     0,0,0
The process stopped, putting back the original instructions Press <enter> to continue

```

```
admin1@linux: ~/Desktop/C|C++
admin1@linux:~/Desktop/C|C++$ sudo ./ptrace 4669
pid    4669
get     H,=,00
put     0,0,0
The process stopped, putting back the original instructions Press <enter> to continue
put     H,=,00
set succeed
detach succeed
admin1@linux:~/Desktop/C|C++$

```

四.Android平台上实现Hook