Notes for playing with ptrace on 64 bits Ubuntu 12.10

Wei Xu / January 12, 2013 / code reading, linux

This blog is the notes during I learning the “Playing with ptrace”(http://www.linuxjournal.com/article/6100).

The original examples was using 32 bits machine, which doesn’t work on my 64 bits Ubuntu 12.10.

Let’s start from the first ptrace example:

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <linux/user.h>   /* For constants
                                   ORIG_EAX etc */
int main()
{   pid_t child;
    long orig_eax;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl("/bin/ls", "ls", NULL);
    }
    else {
        wait(NULL);
        orig_eax = ptrace(PTRACE_PEEKUSER,
                          child, 4 * ORIG_EAX,
                          NULL);
        printf("The child made a "
               "system call %ldn", orig_eax);
        ptrace(PTRACE_CONT, child, NULL, NULL);
    }
    return 0;
}
```

The compiler shows the following error:

```
fatal error: 'linux/user.h' file not found
#include <linux/user.h>
```

Something need to change because of:

1. The ‘linux/user.h’ no longer exists
2. The 64 bits register is R*X, so EAX changed to RAX

There are two solutions to fix this:
1. change ‘linux/user.h’ to ‘sys/reg.h’, and use:

```
long original_rax = ptrace(PTRACE_PEEKUSER, child, 8 * ORIG_RAX, NULL);
```

The addr changed from **‘4 * ORIG_EAX’** to **‘8 * ORIG_RAX’** because it’s the address to read in the user area, and the orig_rax member in user_regs_struct is the **15th** member(start from 0). The definition of ORIG_RAX in file ‘sys/reg.h’ specify it’s position: **# define ORIG_RAX 15**. Because of the other members has size 8 on 64 bits machine, so the addr is: 8 * ORIG_RAX.

The definition of struct user_regs_struct and user in file ‘sys/user.h’:

```
struct user_regs_struct
{
  unsigned long int r15;
  unsigned long int r14;
  unsigned long int r13;
  unsigned long int r12;
  unsigned long int rbp;
  unsigned long int rbx;
  unsigned long int r11;
  unsigned long int r10;
  unsigned long int r9;
  unsigned long int r8;
  unsigned long int rax;
  unsigned long int rcx;
  unsigned long int rdx;
  unsigned long int rsi;
  unsigned long int rdi;
  unsigned long int orig_rax;
  unsigned long int rip;
  unsigned long int cs;
  unsigned long int eflags;
  unsigned long int rsp;
  unsigned long int ss;
  unsigned long int fs_base;
  unsigned long int gs_base;
  unsigned long int ds;
  unsigned long int es;
  unsigned long int fs;
  unsigned long int gs;
};

struct user
{
  struct user_regs_struct   regs;
  int       u_fpvalid;
  struct user_fpregs_struct i387;
  unsigned long int   u_tsize;
  unsigned long int   u_dsize;
  unsigned long int   u_ssize;
  unsigned long int   start_code;
  unsigned long int   start_stack;
  long int     signal ;
  int       reserved;
  struct user_regs_struct*  u_ar0;
  struct user_fpregs_struct*    u_fpstate;
  unsigned long int   magic;
  char      u_comm [32];
  unsigned long int   u_debugreg [8];
};
```

2. change ‘linux/user.h’ to ‘sys/user.h’, and use

```
struct user_regs_struct regs;
ptrace(PTRACE_GETREGS, child, NULL, &regs);
printf ( "The child made a system call %ldn" , regs.orig_rax);
```

The second one is simpler because it doesn’t need to calculate the position, but it read more data than the first one.

I think it would be more clear and easier to understand if we use the address of the orig_rax field directly:

```
struct user* user_space = ( struct user*)0;
long original_rax = ptrace(PTRACE_PEEKUSER, child, &user_space->regs.orig_rax, NULL);
```

We can compile and run it now, but we got: ‘**The child made a system call 59**‘, which is different with ‘**11**‘ from the original post, is there anything wrong? From the file sys/syscall.h, it included file ‘asm/unistd.h’ and the comment says that file list the system calls:

```
/* This file should list the numbers of the system calls the system knows.
   But instead of duplicating this we use the information available
   from the kernel sources.  */
#include <asm/unistd.h>
```

The file ‘asm/unistd.h’ include different files based on __i386__ and __ILP32__:

```
# ifdef __i386__
#  include <asm/unistd_32.h>
# elif defined(__ILP32__)
#  include <asm/unistd_x32.h>
# else
#  include <asm/unistd_64.h>
# endif
```

From the file ‘asm/unistd_64.h’ which contains the system call names for 64 bits machine, we can found that:

```
#define __NR_execve 59
```

Ok, that’s all for the first example, and after understand it, it’s easy to understand the rest parts in part I(http://www.linuxjournal.com/article/6100) and part II(http://www.linuxjournal.com/article/6210).

Share this:

Like this:

Loading...

Wei Xu / January 12, 2013 / code reading, linux

5 thoughts on “Notes for playing with ptrace on 64 bits Ubuntu 12.10”

Pira

March 27, 2013 at 21:41

This is useful for me , thank you!

Romain

October 24, 2013 at 20:03

Got exactly the same problem following the same article (quite obviously), this helps me! Thanks

jiaxi he

August 25, 2016 at 11:14

meet register problem in x64, this help me a lot

Durgesh

September 5, 2016 at 01:28

Nice piece of information, really helped me a lot

sagar suman

August 1, 2018 at 05:14

This article was really helpful. In 64 bit systems function calling convention has been changed(arguments passed to the registers). This should also be addressed in the article.