

Re: Need help in understanding x86 syscall

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/3275.html>

From: Zachary Amsden (zach_at_vmware.com)

Date: 08/11/05

Date: Thu, 11 Aug 2005 12:58:23 -0700
To: Steven Rostedt <rostedt@goodmis.org>

Steven Rostedt wrote:

```
>OK, I get the same on my machine.  
>  
>  
>  
>>On a machine that does not support sysenter, this will give you:  
>>  
>>int $0x80  
>>ret  
>>  
>>The int $0x80 system calls are still fully supported by a sysenter  
>>capable kernel, since it must run older binaries and potentially support  
>>syscalls during early boot up before it is known that sysenter is supported.  
>>  
>>  
>  
>Now is the latest glibc using this. Since I put in a ud2 op in my  
>sysenter_entry code, which is not triggered, as well as an objdump of  
>  
>  
>libc.so shows a bunch of int 0x80 calls.  
>  
>
```

The NPTL version of glibc (the TLS library) uses this.

```
zach-dev2:~ $ ldd /bin/ls  
linux-gate.so.1 => (0xffffe000)  
librt.so.1 => /lib/tls/librt.so.1 (0x4002e000)  
libacl.so.1 => /lib/libacl.so.1 (0x40038000)  
libselinux.so.1 => /lib/libselinux.so.1 (0x4003e000)  
--> libc.so.6 => /lib/tls/libc.so.6 (0x4004c000)  
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x40162000)  
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)  
libattr.so.1 => /lib/libattr.so.1 (0x40174000)
```

Linux–Kernel: Re: Need help in understanding x86 syscall

You'll find getpid much faster with TLS libraries (it's cached, no longer a system call):

With TLS:

```
zach–dev2:Micro–bench $ time ./getpid
```

```
real 0m0.080s
user 0m0.080s
sys 0m0.000s
```

Without TLS:

```
zach–dev:Micro–bench $ time ./getpid
```

```
real 0m5.041s
user 0m2.520s
sys 0m2.520s
```

If you're feeling really masochistic, I've added a demonstration of how you can call sysenter from userspace without glibc. The code verifies that there is no way to exploit the kernel to achieve reading arbitrary memory through a non–flat data segment. It deliberately segfaults at the end. Let me point out this is a very wrong way to do things – you should always use the vsyscall page, and in fact, this code actually depends on the vsyscall page even if it is not apparent. I fake the same frame structure that the vsyscall page would have pushed to simulate a vsyscall entry, but the kernel will always return to the vsyscall page, which then returns back to us. Fun stuff. If you leave the kernel hack for ud2 in your kernel, I would expect it to blow up in amazing fashion when running the code below.

```
zach–dev2:~ $ gcc sysenter.S sysenter.c –o sys
sysenter.c: In function `main':
sysenter.c:34: warning: passing arg 2 of `signal' from incompatible
pointer type
sysenter.c:49: warning: passing arg 3 of `sysenter_call_2' makes pointer
from in
teger without a cast
sysenter.c:22: warning: return type of `main' is not `int'
zach–dev2:~ $ ./sys
interrupted %ebp = 0xbaadf00d
pew
Segmentation fault (core dumped)
zach–dev2:~ $ gdb sys core
GNU gdb 6.2.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i586–suse–linux"...Using host libthread_db
```

Re: Need help in understanding x86 syscall

Linux–Kernel: Re: Need help in understanding x86 syscall

```
library "  
/lib/tls/libthread_db.so.1".
```

```
Core was generated by `./sys'.  
Program terminated with signal 11, Segmentation fault.
```

```
warning: current_sos: Can't read pathname for load map: Input/output error
```

```
Reading symbols from /lib/tls/libc.so.6...done.  
Loaded symbols for /lib/tls/libc.so.6  
Reading symbols from /lib/ld–linux.so.2...done.  
Loaded symbols for /lib/ld–linux.so.2  
#0 0xffffe410 in ?? ()  
(gdb) print $eax  
$1 = -14  
(gdb)
```

```
#define EFAULT 14 /* Bad address */
```

```
int main(int argc, char *argv[]) {  
    int j;  
    for (j = 0; j < 1000000; j++) {  
        getpid(); getpid(); getpid(); getpid(); getpid();  
        getpid(); getpid(); getpid(); getpid(); getpid();  
    }  
}
```

```
#include <sys/syscall.h>
```

```
.text  
.global sysenter_call  
.global sysenter_call_2
```

```
/* void sysenter_call(pid_t pid, int signo, short ds, void *addr) */
```

```
sysenter_call:  
    push %ebx  
    push %edi  
    push %ebp  
    push %ds  
    movl %esp, %edi  
    movl 20(%esp), %ebx /* pid */  
    movl 24(%esp), %ecx /* signo */  
    movl 28(%esp), %ds /* exploit DS */  
    movl 32(%esp), %ebp
```

Linux–Kernel: Re: Need help in understanding x86 syscall

```
movl %ebp, %esp
push $sysenter_return
push %ecx
push %edx
subl $16, %ebp
push $0xbaadf00d
movl $SYS_kill, %eax
sysenter
```

/* vsyscall page will ret to us here */

sysenter_return:

```
mov %edi, %esp
pop %ds
pop %ebp
pop %edi
pop %ebx
ret
```

sysenter_call_2:

```
push %ebx
push %ebp
movl 12(%esp), %ebx /* pid */
movl 16(%esp), %ecx /* signo */
movl 20(%esp), %ebp
movl $SYS_kill, %eax
sysenter
```

.data

test: .long 0

```
#include <stdio.h>
#include <signal.h>
#include <asm/ldt.h>
#include <asm/segment.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/mman.h>
#define __KERNEL__
#include <asm/page.h>
```

```
extern void sysenter_call(pid_t pid, int signo, short ds, void *addr);
extern void sysenter_call_2(pid_t pid, int signo, void *addr);
void catch_sig(int signo, struct sigcontext ctx)
{
    __asm__ __volatile__ ("mov %0, %%ds" : : "r" (__USER_DS));
    printf("interrupted %%ebp = 0x%x\n", ctx.ebp);
    if (ctx.ebp == 0xbaadf00d)
        printf("phew\n");
}
```

Linux–Kernel: Re: Need help in understanding x86 syscall

```
}  
  
void main(void)  
{  
    struct user_desc desc;  
    short ds;  
    unsigned long addr;  
    unsigned *stack;  
    unsigned long offset;  
  
    stack = (unsigned *)mmap(0, 4096, PROT_EXEC|PROT_READ|PROT_WRITE,  
        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);  
    stack = &stack[1024];  
    addr = 0xf0000;  
    offset = __PAGE_OFFSET-(unsigned)stack+addr+16;  
    signal(SIGUSR1, catch_sig);  
    desc.entry_number = 0;  
    desc.base_addr = offset;  
    desc.limit = 0xffffffff;  
    desc.seg_32bit = 1;  
    desc.contents = MODIFY_LDT_CONTENTS_DATA;  
    desc.read_exec_only = 0;  
    desc.limit_in_pages = 1;  
    desc.seg_not_present = 0;  
    desc.useable = 1;  
    if (modify_ldt(1, &desc, sizeof(desc)) != 0) {  
        perror("modify_ldt");  
    }  
    ds = 0x7; /* TI | RPL 3 */  
    sysenter_call(getpid(), SIGUSR1, ds, stack);  
    sysenter_call_2(getpid(), SIGSTOP, __PAGE_OFFSET+4096);  
    printf("not reached – core should show %%eax == -EFAULT\n");  
}
```

–
To unsubscribe from this list: send the line "unsubscribe linux–kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo–info.html>
Please read the FAQ at <http://www.tux.org/lkml/>