vsyscall and vDSO

Adrien « schischi » Schildknecht

March 17, 2014

Section 1

Introduction

Goal

Adrien « schischi » Schildknecht

In computing, a system call is how a program requests a service from an operating system's kernel. This may include hardware related services (e.g. accessing the hard disk), creating and executing new processes, and communicating with integral kernel services (like scheduling). System calls provide an essential interface between a process and the operating system.

How to speed it up?

The kernel maps into user space a page (rw) containing some variable and the implementation of some syscalls.

- The syscall is executed in userland
- Reduce the overhead of context switching
- Variables are updated on each clock interrupt
- Only apply to « safe » syscalls (only read a variable)
  - gettimeofday()
  - time()
  - getcpu()

vsyscall.h


```
1  #ifndef _ASM_X86_VSYSCALL_H
   #define _ASM_X86_VSYSCALL_H
3
   enum vsyscall_num {
5     __NR_vgettimeofday,
      __NR_vtime,
7     __NR_vgetcpu,
   };
9
   #define VSYSCALL_START (-10UL << 20)
11 #define VSYSCALL_SIZE 1024
   #define VSYSCALL_END (-2UL << 20)
13 #define VSYSCALL_MAPPED_PAGES 1
   #define VSYSCALL_ADDR(vsyscall_nr) (VSYSCALL_START+
       VSYSCALL_SIZE*(vsyscall_nr))
15
17 #endif /* _ASM_X86_VSYSCALL_H */
```

/usr/include/asm/vsyscall.h

Section 2

vsyscall

Limitations

- small memory allocated
- only 4 system calls
- statically allocated (nailed down in the kernel ABI)
  - last page - 1 (fixed address)
  - security flaw
- structure of an elf ??

Security

- Static, no ASLR
- several machine instructions that invoke syscalls (ROP)
- execute flag on a page with some variables. Thus allowing to jump into when it held the right value

Security patch

- not used anymore in 32bits (archlinux 3.13, glibc 2-19)
- variables have been moved into a separate page with execute permission turned off

```
sh> cat linux-3.13.5/arch/x86/kernel/vsyscall_64.c
void __init map_vsyscall(void)
{
    ...
    __set_fixmap(VSYSCALL_FIRST_PAGE,
    physaddr_vsyscall,
                vsyscall_mode == NATIVE
                ? PAGE_KERNEL_VSYSCALL
                : PAGE_KERNEL_VVAR);
    __set_fixmap(VVAR_PAGE, physaddr_vvar_page,
    PAGE_KERNEL_VVAR);
}

```

Section 3

vDSO

Definition

- A memory area allocated in user space which exposes some kernel functionalities at user space in a safe manner.
- A shared object exposed by the kernel
- Dynamically allocated
- can have more than 4 system calls.

```
sh> ldd ./a.out
    linux-vdso.so.1 (0x000...)
    # linux-gate.so.1 => (0x000...)
    libc.so.6 => /usr/lib/libc.so.6 (0x000...)
    /lib64/ld-linux-x86-64.so.2 (0x000...)

```

Section 4

Analasys

Subsection 1

32bits

vsyscall


```
1 sh> cat /proc/self/maps
  ...
3
```

No [vsyscall] present !

32bits vDSO


```
gdb> info proc map
gdb> dump binary memory vdso_32.dump ...

sh> file vdso_32.dump
vdso_32: ELF 32-bit LSB shared object, Intel 80386

sh> objdump -T vdso_32.dump
ffffe414 ... __kernel_vsyscall

```

__kernel_vsyscall is a method determined at boot time to make a system call using the fastest available method (sysenter, syscall, int 0x80, ...)

```
sh> ls linux/arch/x86/vdso/vdso32/*.S

```

32bits glibc


```
sh> objdump -d /lib/libc-2.19.so | fgrep -A12 '<time>:'

000a70d0 <time>:
 a70d0:  31 c0                   xor    %eax,%eax
 a70d2:  8b 54 24 04             mov    0x4(%esp),%edx
 a70d6:  89 c1                   mov    %eax,%ecx
 a70d8:  87 cb                   xchg   %ecx,%ebx
 a70da:  b8 0d 00 00 00          mov    $0xd,%eax
 a70df:  65 ff 15 10 00 00 00    call   *%gs:0x10
                                 # located in the vDSO
 a70e6:  87 cb                   xchg   %ecx,%ebx
 a70e8:  85 d2                   test   %edx,%edx
 a70ea:  74 02                   je     a70ee <time+0x1e>
 a70ec:  89 02                   mov    %eax,(%edx)
 a70ee:  f3 c3                   repz ret
```

Adrien « schischi » Schildknecht

Calling time() in a 32bits linux program :

- call time() from glibc
- call __kernel_vsyscall from the vDSO
- sysenter, syscall or 0x80
- syscall is executed in kernel space. . .

Subsection 2

64bits

vDSO


```
gdb> info proc map
gdb> dump binary memory vdso_64.dump ...

sh> file vdso_64.dump
vdso_64: ELF 64-bit LSB shared object, x86-64

sh> objdump -T vdso_64.dump
ffffffffff700870  clock_gettime
ffffffffff700870  __vdso_clock_gettime
ffffffffff700b70  gettimeofday
ffffffffff700b70  __vdso_gettimeofday
ffffffffff700d20  getcpu
ffffffffff700d20  __vdso_getcpu
ffffffffff700d00  time
ffffffffff700d00  __vdso_time
```

64bits vsyscall


```
gdb> dump binary memory vsyscall_64.dump
     ffffffffff600000 ffffffffff601000
sh> file vsyscall_64.dump
vsyscall_64: data # wtf (not an elf !)
sh> ndisasm vsyscall_64.dump | less
00000000  48                      dec ax
00000001  C7C06000                mov ax,0x60
00000005  0000                    add [bx+si],al
00000007  0F05                    syscall
00000009  C3                      ret
0000000A  CC                      int3
0000000B  CC                      int3
# a lot of int3
sh> ndisasm vsyscall_64 | grep mov
00000001  C7C06000 mov ax,0x60   # gettimeofday
00000401  C7C0C900 mov ax,0xc9   # time
00000801  C7C03501 mov ax,0x135  # getcpu

```

The code in the vsyscall page has been removed and replaced by a special trap instruction.

The kernel emulate the desired syscall in kernel space.

The page contain a method which use a kernel syscall, emulating a virtual system call.

Vsyscalls are now slower than standard syscalls. . . But they should not be used anyway. . .

Retrieve time symbol from vDSO. Substitute with vsyscall if not available.

```
 p = _dl_vdso_vsym ("__vdso_getcpu", &linux26);
  /* If the vDSO is not available we fall back on
    the old vsyscall.  */
#define VSYSCALL_ADDR_vgetcpu    0xffffffffff600800
  if (p == NULL)
    p = (void *) VSYSCALL_ADDR_vgetcpu;

```

The end

The end.