


```

3    movq    %fs:40, %rax           Retrieve canary
4    movq    %rax, 8(%rsp)         Store on stack
5    xorl    %eax, %eax           Zero out register
6    movq    %rsp, %rdi           Compute buf as %rsp
7    call    gets                 Call gets
8    movq    %rsp, %rdi           Compute buf as %rsp
9    call    puts                 Call puts
10   movq    8(%rsp), %rax         Retrieve canary
11   xorq    %fs:40, %rax         Compare to stored value
12   je      .L9                  If =, goto ok
13   call    __stack_chk_fail     Stack corrupted!
14   .L9:                               ok:
15   addq    $24, %rsp            Deallocate stack space
16   ret

```

这个版本的函数从内存中读出一个值(第3行),再把它存放在栈中相对于`%rsp`偏移量为8的地方。指令参数`%fs:40`指明金丝雀值是用段寻址(segmented addressing)从内存中读入的,段寻址机制可以追溯到80286的寻址,而在现代系统上运行的程序中已经很少见到了。将金丝雀值存放在一个特殊的段中,标志为“只读”,这样攻击者就不能覆盖存储的金丝雀值。在恢复寄存器状态和返回前,函数将存储在栈位置处的值与金丝雀值做比较(通过第11行的`xorq`指令)。如果两个数相同,`xorq`指令就会得到0,函数会按照正常的方式完成。非零的值表明栈上的金丝雀值被修改过,那么代码就会调用一个错误处理例程。

栈保护很好地防止了缓冲区溢出攻击破坏存储在程序栈上的状态。它只会带来很小的性能损失,特别是因为GCC只在函数中有局部`char`类型缓冲区的时候才插入这样的代码。当然,也有其他一些方法会破坏一个正在执行的程序的状态,但是降低栈的易受攻击性能够对抗许多常见的攻击策略。

 **练习题 3.48** 函数`intlen`、`len`和`iptoa`提供了一种很纠结的方式,来计算表示一个整数所需要的十进制数字的个数。我们利用它来研究GCC栈保护者措施的一些情况。

```

int len(char *s) {
    return strlen(s);
}

void iptoa(char *s, long *p) {
    long val = *p;
    sprintf(s, "%ld", val);
}

int intlen(long x) {
    long v;
    char buf[12];
    v = x;
    iptoa(buf, &v);
    return len(buf);
}

```

下面是`intlen`的部分代码,分别由带和不带栈保护者编译: