坏的状态，试图重新加载寄存器或执行 ret 指令时，就会出现很严重的错误。

一种特别常见的状态破坏称为缓冲区溢出(buffer overflow)。通常，在栈中分配某个字符数组来保存一个字符串，但是字符串的长度超出了为数组分配的空间。下面这个程序示例就说明了这个问题：

```
/* Implementation of library function gets() */
char *gets(char *s)
{
    int c;
    char *dest = s;
    while ((c = getchar()) != '\n' && c != EOF)
        *dest++ = c;
    if (c == EOF && dest == s)
        /* No characters read */
        return NULL;
    *dest++ = '\0'; /* Terminate string */
    return s;
}

/* Read input line and write it back */
void echo()
{
    char buf[8];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

前面的代码给出了库函数 gets 的一个实现，用来说明这个函数的严重问题。它从标准输入读入一行，在遇到一个回车换行字符或某个错误情况时停止。它将这个字符串复制到参数 s 指明的位置，并在字符串结尾加上 null 字符。在函数 echo 中，我们使用了 gets，这个函数只是简单地从标准输入中读入一行，再把它回送到标准输出。

gets 的问题是它没有办法确定是否为保存整个字符串分配了足够的空间。在 echo 示例中，我们故意将缓冲区设得非常小——只有 8 个字节长。任何长度超过 7 个字符的字符串都会导致写越界。

检查 GCC 为 echo 产生的汇编代码，看看栈是如何组织的：

```
    void echo()
1   echo:
2     subq    $24, %rsp          Allocate 24 bytes on stack
3     movq    %rsp, %rdi         Compute buf as %rsp
4     call    gets               Call gets
5     movq    %rsp, %rdi         Compute buf as %rsp
6     call    puts               Call puts
7     addq    $24, %rsp          Deallocate stack space
8     ret                        Return
```

图 3-40 画出了 echo 执行时栈的组织。该程序把栈指针减去了 24(第 2 行)，在栈上分配了 24 个字节。字符数组 buf 位于栈顶，可以看到，%rsp 被复制到%rdi 作为调用 gets 和 puts 的参数。这个调用的参数和存储的返回指针之间的 16 字节是未被使用的。只要用户输入不超过 7 个字符，gets 返回的字符串(包括结尾的 null)就能够放进为 buf 分配的