


然而，一个执著的攻击者总是能够用蛮力克服随机化，他可以反复地用不同的地址进行攻击。一种常见的把戏就是在实际的攻击代码前插入很长一段的 `nop` (读作“no op”，no operation 的缩写) 指令。执行这种指令除了对程序计数器加一，使之指向下一条指令之外，没有任何的效果。只要攻击者能够猜中这段序列中的某个地址，程序就会经过这个序列，到达攻击代码。这个序列常用的术语是“空操作雪橇(nop sled)” [97]，意思是程序会“滑过”这个序列。如果我们建立一个 256 个字节的 `nop sled`，那么枚举 $2^{15} = 32\,768$ 个起始地址，就能破解 $n = 2^{23}$ 的随机化，这对于一个顽固的攻击者来说，是完全可行的。对于 64 位的情况，要尝试枚举 $2^{24} = 16\,777\,216$ 就有点儿令人畏惧了。我们可以看到栈随机化和其他一些 ASLR 技术能够增加成功攻击一个系统的难度，因而大大降低了病毒或者蠕虫的传播速度，但是也不能提供完全的安全保障。

 **练习题 3.47** 在运行 Linux 版本 2.6.16 的机器上运行栈检查代码 10 000 次，我们获得地址的范围从最小的 `0xfffffb754` 到最大的 `0xfffffd754`。

- A. 地址的大概范围是多大？
- B. 如果我们尝试一个有 128 字节 `nop sled` 的缓冲区溢出，要想穷尽所有的起始地址，需要尝试多少次？

2. 栈破坏检测

计算机的第二道防线是能够检测到何时栈已经被破坏。我们在 `echo` 函数示例(图 3-40)中看到，破坏通常发生在当超越局部缓冲区的边界时。在 C 语言中，没有可靠的方法来防止对数组的越界写。但是，我们能够在发生了越界写的时候，在造成任何有害结果之前，尝试检测到它。

最近的 GCC 版本在产生的代码中加入了一种栈保护者(stack protector)机制，来检测缓冲区越界。其思想是在栈帧中任何局部缓冲区与栈状态之间存储一个特殊的金丝雀(canary)值^①，如图 3-42 所示 [26, 97]。这个金丝雀值，也称为哨兵值(guard value)，是在程序每次运行时随机产生的，因此，攻击者没有简单的办法能够知道它是什么。在恢复寄存器状态和从函数返回之前，程序检查这个金丝雀值是否被该函数的某个操作或者该函数调用的某个函数的某个操作改变了。如果是的，那么程序异常中止。

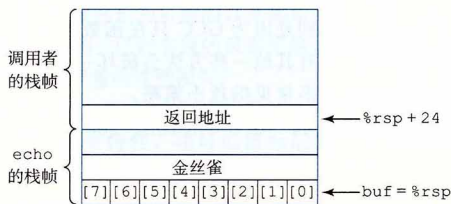


图 3-42 `echo` 函数具有栈保护者的栈组织(在数组 `buf` 和保存的状态之间放了一个特殊的“金丝雀”值。代码检查这个金丝雀值，确定栈状态是否被破坏)

最近的 GCC 版本会试着确定一个函数是否容易遭受栈溢出攻击，并且自动插入这种溢出检测。实际上，对于前面的栈溢出展示，我们不得不用命令行选项“`-fno-stack-protector`”来阻止 GCC 产生这种代码。当不用这个选项来编译 `echo` 函数时，也就是允许使用栈保护者，得到下面的汇编代码：

```
void echo()
1  echo:
2  subq    $24, %rsp           Allocate 24 bytes on stack
```

① 术语“金丝雀”源于历史上用这种鸟在煤矿中察觉有毒的气体。