中的一个漏洞，接着发现如何利用该漏洞攻击计算机。

每一种攻击都涉及特定程序中的特定漏洞，其中利用某些反复出现的漏洞展开的攻击值得我们学习。在本节中，我们将研究一些攻击的工作原理，由于本书的核心是操作系统，因此重点将放在如何攻击操作系统上，而利用系统和软件漏洞对网页和数据库的攻击方式本节都没有涉及。

有很多方式可以对漏洞进行利用，在一种直接的方法中，攻击者会启动一个脚本，该脚本按顺序进行如下活动：

1) 运行自动端口扫描，以查找接受远程连接的计算机。

2) 尝试通过猜测用户名和密码进行登录。

3) 一旦登录成功，则启动特定的具有漏洞的程序，并产生输入使得程序中的漏洞被触发。

4) 如果该程序运行SETUID到root，则创建一个SETUID root shell。

5) 启动一个僵尸程序，监听IP端口的指令。

6) 对目标机器进行配置，确保该僵尸程序在系统每次重新启动后都会自动运行。

上述脚本可能会运行很长时间，但是它有很可能最终成功。攻击者确保只要目标计算机重新启动时，僵尸程序也启动，就使得这台计算机一直被控制。

另一种常用的攻击方式利用了已经感染病毒的计算机，在该计算机登录到其他机器的时候，计算机中的病毒启动目标机器中的漏洞程序（就像上面提到的脚本一样）。基本上只有第一步和第二步与上述脚本文件不同，其他步骤仍然适用。不论哪种方法，攻击者的程序总是要在目标机器中运行，而该机器的所有者对该恶意程序一无所知。

9.6.1 缓冲区溢出攻击

之所以有如此多的攻击是因为操作系统和其他应用程序都是用C语言写的（因为程序员喜欢它，并且用它来进行有效的编译）。但遗憾的是，没有一个C编译器可以做到数组边界检查。如下面的代码虽然并不合法，但系统却没有进行检验：

```
int i;
char c[1024];
i=12000;
c[i]=0;
```

结果内存中有10 976个字节超出了数组c的范围，并有可能导致危险的后果。在运行时没有进行任何检查来避免这种情况。

C语言的属性导致了下列攻击。在图9-24a中，我们看到主程序在运行时局部变量是放在栈里的。在某些情况下，系统会调用过程A，如图9-24b所示。标准的调用步骤是把返回地址（指向调用语句之后的指令）压入栈，然后将程序的控制权交给A，由A不断减少栈指针地址来分配本地变量的存储空间。

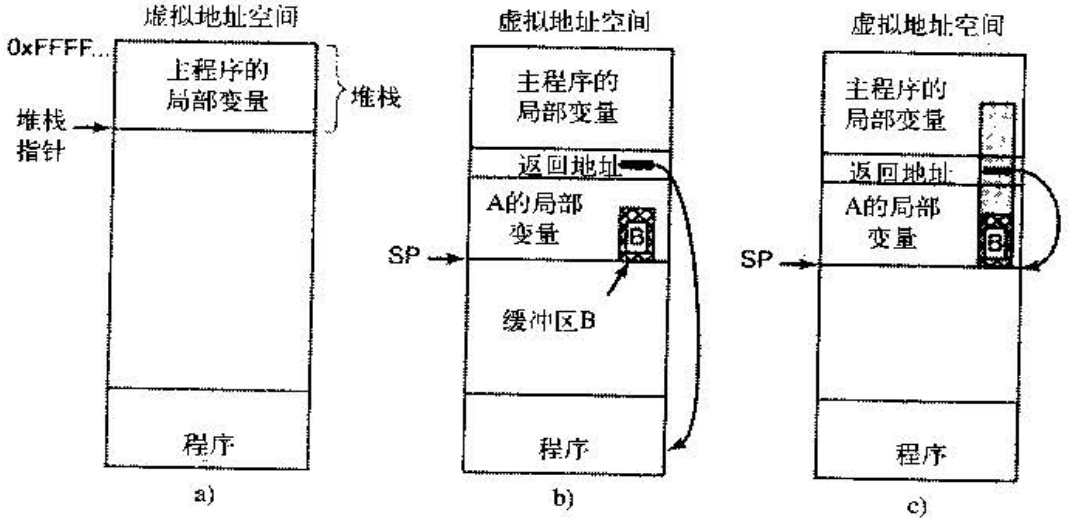

图9-24 a) 主程序运行时的情况；b) 调用过程A后的情况；c) 灰色字体表示的缓冲溢出