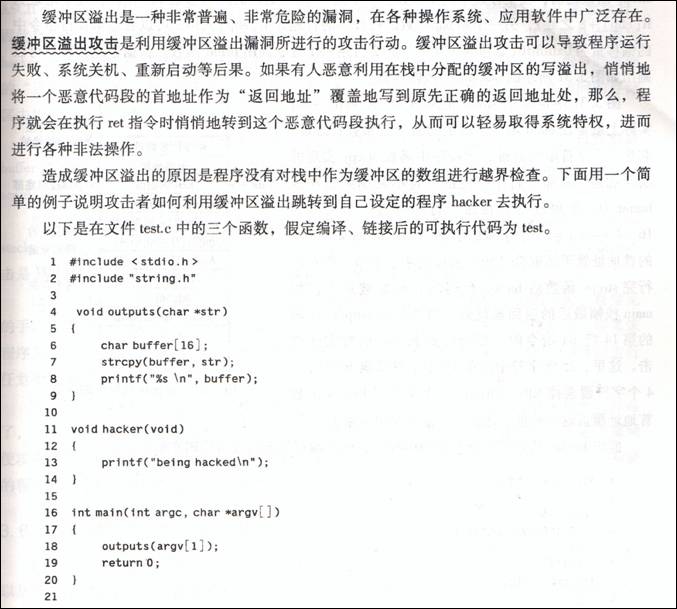
# 缓冲区漏洞分析及解决方案

黄郭斌 计科1502班 201507010206



**·缓冲区漏洞**：

由于strcpy函数并不提供对输入字符串的长度检查(the function is dangerous)，而是无论从client获得多少长字符串的，直接将读到的str写入栈中的临时数组buffer中，这样一来，攻击者可以通过预测栈空间的大小，通过控制str的长度和内容，使得写入buffer之后，溢出buffer开辟的正常空间，修改当前栈帧顶部的返回地址，将其修改为攻击代码的起始地址，从而实现对系统的控制。

**·解决方案**：

1. **使用更安全的函数**

如使用含有长度检查的strncpy代替strcpy（前者发现source string长度长于dest str的长度限制n，则只会复制前n个字符），这样，系统拥有了保证数组长度不超过某一个特定值的主动权。

1. **栈随机化**

由于在系统中插入攻击代码，攻击者不但要插入代码，还需要插入指向这段代码的指针，由于攻击代码存在在栈区的buffer中，而在过去，使用同样的程序和操作系统，不同的机器栈的位置是相当固定的，因此，攻击者如果可以确定一个常见的web服务器所使用的栈空间，就可以在许多机器上实施攻击。

通过栈随机化，也就是通过一个函数alloca在栈首分配一个随机数量的空间（每次分配的空间都不一样），这样，攻击者将无法预测攻击代码的起始地址（但是通过“空操作雪橇”，还是可以强力破解栈随机化，不过随着栈首分配的空间随机范围的增大，攻击成功的可能性将大大减小）。

1. **栈破坏检测**

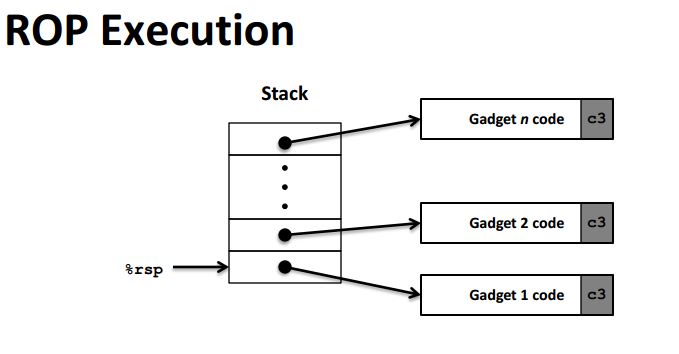
通过在buffer和返回地址之间划出一个专门的区间，存放一个每次创建函数栈帧都不同且随机的“金丝雀值”（这样攻击者就不能提前预测它的值是什么），来阻止发生缓冲区溢出时候可能导致的不可预测的结果。具体的做法是，在恢复寄存器状态并返回之前，检测一下之前设置的“金丝雀”值与之前设置的值是否发生了变化，如果是，那么代码会调用一个点错误处理进程，并提前终止函数，阻止其返回。

1. **限制可执行代码的区域**

通过设置系统对虚拟存储器空间内不同区段的访问形式（比如：是否可写，是否可读），加入一个是否可执行的标志位（需要系统或硬件支持），保证在栈区段的数据无法被执行，即可防止执行攻击代码。

**Return Oriented Programming Attack （ROP）**

但是，现在有一些攻击代码是通过拼凑一些代码段ret命令前的指令（甚至通过特定的地址，完全可能将一个原先5字节长的指令解读成一个4字节的完全不同的指令，只要指向该指令第二个字节即可），从而拼凑出攻击者所希望是实现的程序，如下图：

****

通过缓冲区溢出修改最底位的返回地址（注意：最低位的返回地址之上的内存依然被攻击者覆盖为一系列攻击代码片段的起始地址），在函数返回之后，跳转到如图Gadget 1 code（位于代码段，或者链接进来的库函数片段）执行，执行完，返回，自然而然又跳转到了Gadget 2 code执行，从而实现对目标系统的攻击。

**注意：**

1.由于攻击者利用的代码均为程序自身的代码，其地址有别与栈，一般位置是确定的，因此可以很容易被攻击者预测出，栈随机化策略失效。

2.这些代码存在的区域一般为代码段和链接的库函数段，因此必然是可执行的，限制可执行代码区域的策略失效。

3.所幸，即使是通过**ROP，也依然无法战胜栈破坏检测**。因此，通过编译器在有风险的区域加入stack protector，完全可以防止缓冲区溢出攻击。（据说目前无人能破解栈破坏检测）