

在程序运行过程中，内存基本可以分为这四部分:

1~栈区（Stack）用于存储局部变量和函数调用信息，具有大小限制，管理方式为后进先出，这个一般都有限制，linux下可用limit指令查看

2~堆区（Heap）用于动态分配内存，程序可以在运行时根据需要请求内存，管理相对灵活，但需要手动释放。就比如new,malloc,calloc运算符申请的内存就是在这个区申请的

3~数据段（Data）用于存储静态分配的数据，包括全局变量、静态变量和字符串常量，这些数据在程序启动时分配内存。

4~文本段（Text）存储可执行的机器指令，通常为只读，包含程序的代码和共享库。就比如链接的时候使用的一些库函数就储存在这个内存空间中

对于一个数据，通过观察程序给他们分配内存时的不同位置，一般就能看出这个数据的不同性质

简单说一下缓冲区溢出的情况，对于一个函数的缓冲区，其中包括系统自动生成的缓冲区和函数自定义的等效的数组等的缓冲区，在进行数据的输入是时，程序会先将数据送到缓冲区，然后才会将数据读入到其对应的内存区域中，对于一些程序，其并不会检查输入的数据是否超过缓冲区的大小，对于一些程序，可能恶意利用这个特性，对程序进行恶意注入代码实现病毒等操作

对于避免关于缓冲区的这种攻击的几种操作：  
1~使用更加安全的函数，比如用fgets代替gets等，这种函数会检查输入的数据是否会修改缓冲区以外的数据，比如fgets多的一个参数可以来指示输入数据应该占多少，若超过了这个数，会对之后的字符进行截断

2~ASLR **地址空间布局随机化** 使得程序在每次运行时对栈，堆内存区的起始地址都进行随机分配，还有一些共享库的地址也会随机化

3~对栈中内存中的位置进行标志，可实现对未知代码（可能是注入的）进行忽略，使得其发挥不了作用

4~**堆栈保护器**通过在一个函数栈中插入一些随机(金丝雀)值，在函数结束后对每个金丝雀值进行识别是否被篡改来实现对程序的监控，一般在函数返回地址和缓冲区之间会有一个金丝雀值来监控

此时再来说一下由保护的升级而引来的面向返回编程：

通过利用现有的代码片段和返回语句的特性来进行组合以实现自己所期望的功能，其中的每一个目标代码片段就是一个gadget，每一个一般都是一段代码中的一小段，通过注入指令对应的地址的返回语句使得程序跳到对应的代码片段，并使得在该片段执行完过后能够再次跳转到目标gadget，不断跳转来实现目标功能

攻击中将这些gadget用返回语句来实现一个ROP链，并通过缓冲区注入这个ROP链来进行攻击

最后简单介绍下联合体，这个在内存中依据最大的数据成员所占内存来进行内存的分配，在对联合体的成员中进行赋值时，一般只会对最后一个数据进行处理，因为所有成员都共享一块内存空间，对一个成员赋值会影响所有的成员值，所以对之前的成员进行访问其实是没有意义的