栈中的守护天使--GS，亦称作Stack Canary / Cookie，从VS2003起开始启用（也就说，GS机制是由编译器决定的，跟操作系统无关），是堆栈溢出中最令人头疼的一道关卡

GS机制分三个步骤：计算随机种子 --> canary写入栈帧 --> GS校验

[1]程序启动时，读取**.data**的第一个**DWORD**作为基数，然后和各种元素（时间戳，进程ID，线程ID，计数器等等）进行XOR加密

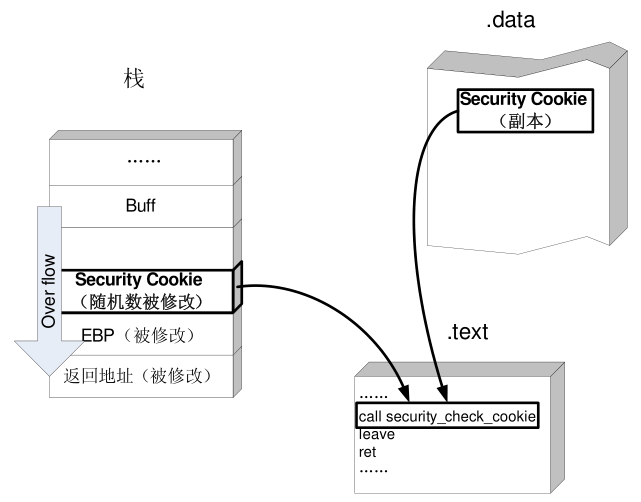
[2]然后将加密后的种子再次写入**.data**的第一个**DWORD**

[3]函数在执行前，把加密后的种子取出，与当前**esp**进行异或计算，结果存入EBP的前面

[4函数主体正常执行。

[5]**函数返回前**（retn前一点），把cookie取出与esp异或计算后，调用\_\_security\_check\_cookie函数进行检查，与.data节里的种子进行比较，

如果校验通过，则返回原函数继续执行；如果校验失败，则程序终止。



然而，估计大家也发现了，为何Buff和Cookie之间还有些空隙？其实......在旧版本的编译器里，局部变量是随机摆放的（指针，int，字符串位置随机），

所以这里就还存在一丝安全隐患->\_->那就是Buff可能在不压过Cookie的情况下覆盖一些局部变量，

所以，后期的编译器就推出了--**变量重排技术**（VS2005之后的优化编译选项）



在编译时根据**局部变量的类型**对变量在栈帧中的位置进行调整，将字符串变量移动到栈帧的**高地址**。这样可以防止该字符串溢出时破坏其他的局部变量。

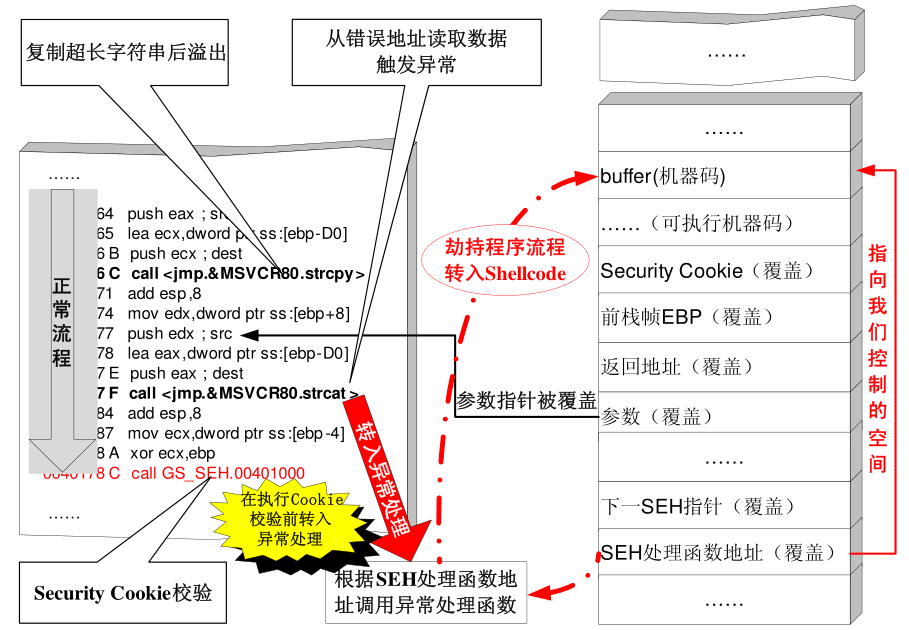
同时还会将指针参数和字符串参数复制到内存中低地址（如图中的arg），防止函数参数被破坏

绕过方法一：攻击S.E.H

GS 机制并没有对 S.E.H 提供保护，即异常处理函数的地址在cookie之前。

我们首先通过超长字符串覆盖掉异常处理函数指针，然后想办法触发一个异常，程序就会转入异常处理，由于异常处理函数指针已经被我们覆盖，

那么我们就可以通过劫持S.E.H来控制程序的后续流程......



这种绕过技术虽然简单且有效，但随后推出的SafeSEH以及SEHOP，都会将此方法封杀

绕过方法二：Leak Canary

可以通过printf（格式化字符串漏洞）泄露，但要注意，cookie一般从00开始，而printf会00截断，

所以用此方法时要越过前两位

绕过方法三：Overwrite Cnary

即同时覆盖栈中的cookie及.data中的cookie，额......这种思路是好，但实际上几乎不可能，因为cookie存在

TLS（即线程局部存储）中，而TLS地址是被随机化，故除非一些特定场景，很难实现

综上所述，GS机制绕过是艰难的，必须结合实际场景运用不同的方法，其中第二种方法用的比较多，

大家可以多多研究，之后有新思路推出时，我再及时更新^\_^

祝大家生活愉快~~