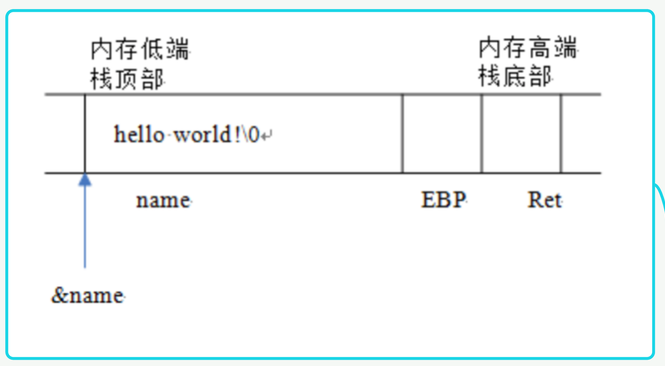
缓冲区溢出攻击及防御技术

- 缓冲区溢出原理
  - 能被攻击原因
    - C语言对数据和指针不自动进行边界检测，一些字符串处理函数strcpy、sprintf等存在严重的安全问题
    - 程序员代码写的不严谨
    - 返回地址放在堆栈的底部，可以通过溢出覆盖放回地址
    - 堆栈的属性一般是可执行的，是的而已代码得意执行
  - 防御
    - 提取用于攻击的shellcode的普遍特征作为攻击特征，过滤掉这样的数据包，或者触发报警
    - 对特定的服务限定请求数据的值和范围，比如，某一服务器要求请求数据为可打印字符串，如果发现这一服务的请求存在不可打印字符则认为发生攻击
    - 源码BUG查找，使用安全的函数
    - 运行期保护
      - 数据边界检查
      - 保证放回指针的完整性
    - 加强系统保护
      - 保护系统信息
      - 关闭不需要的服务
      - 最小权限原则
      - 使用系统的堆栈补丁
      - 检查系统漏洞，及时打上补丁
    - 使用类型安全的语言开发
  - 栈溢出
    - 在堆栈中压入的数据超过预先给堆栈分配的容量。
    - 关注数据区和堆栈区
      - 使用栈
        - ESP 栈顶指针
          - 随着数据入栈出栈发生变化
        - EBP 基地址指针
          - 通过BP加上偏移地址，可以方便的引用函数参数已经局部变量
      - 局部变量发生溢出，很有可能会覆盖掉EBP设置RET（返回地址）
    - 首先把指令寄存器（CPU要执行的下一条指令地址）压入栈，作为程序的返回地址（RET），之后作为栈的基址寄存器（EBP）它指向当前函数栈针的地步，而后把当前的栈顶指针，ESP拷贝到EBP基址寄存器，作为新的基地址。
    - 一段连续内存块
      - 代码段
        - 存放程序的机器码和只读数据，可执行指令从这里取得
      - 数据段
        - 以初始化的数据（.data）
          - 保存全局和静态已初始化的变量
        - 未初始化的数据（.bss）
          - 全局和静态未初始化的变量
      - 堆栈端
        - 堆
          - 位于BBS内存段的上边，用来存储运行时分配的变量
          - 动态扩张或缩减
          - malloc()、new()创建
          - delete()删除
        - 栈
          - 存储函数调用时临时信息结构
          - 编译器需要的时候分配，不需要的时候自动清除
          - 先进后出队列
  - 堆溢出
    - 不如栈溢出流行
      - 比栈溢出难度更大
      - 需要结合其他技术
      - 对于内存中变量的组织方式有一定的要求
  - BBS溢出
  - 格式化串溢出
  - 攻击三部曲
    - 构造需要运行的shellcode，并将其放到目标系统的内存
    - 获得缓存区的大小和定位溢出点RET的位置
    - 控制程序跳转，改变程序流程
  - 缓冲区溢出攻击
    - 特点
      - 不需要太多的先决条件
      - 杀伤力很强
      - 技术性强
    - 破坏性
      - 极其容易使服务停止运行
    - 隐蔽性
      - 漏洞发现者并非编写者
      - 攻击代码很短
- 方法
  - 在程序的地址空间安排适当代码
  - 使控制流跳转到攻击代码
    - 函数指针
    - 激活记录
      - 当函数调用时，堆栈中会留驻一个Activation Record 包含函数返回的地址。溢出修改这一记录。
    - 长跳转缓存区
      - C语言的 setjmp 与 longjmp
- 缓冲区
  - 相同数据类型实例的一个连续的计算机内存块，用于保存数据。
- 溢出
  - 所填充的数据超出了原有的边界
- 代码植入技术
  - shellcode
    - 核心部分，能完成特殊任务的二进制代码
  - 返回地址
    - shellcode的入口地址
    - 设法用shellcode的入口地址覆盖某个跳转指令
  - 填充数据
    - 提高shellcode命中率，在前面安排一定的填充数据
  - 模式
    - 参数
      - S 代表 Shellcode
      - R 代表返回地址
      - N 代表填充数据
      - A 代表环境变量
    - NSR
    - RNS
    - AR



思维导图

```
                buf1            间距            buf2
覆盖前：[xxxxxxxxxxxxxxxx] [xxxxxxxx] [aaaaaaaaaaaaaaaa]
    低址- - - - - - - - - - - - - - - - - - - - - - > 高址
覆盖后：[bbbbbbbbbbbbbbbb] [bbbbbbbb] [bbbbbbbbaaaaaaaa]
```