对抗缓冲区溢出攻击（选读）

缓冲区溢出是一种常见的安全漏洞，可能导致程序崩溃、数据泄露或恶意代码的执行。因此，对于程序员、编译器和操作系统来说，防范缓冲区溢出至关重要。本章将从这三个方面介绍缓冲区溢出的防范技术，供对此感兴趣的同学参考。

1 从程序员角度去防范

- **避免使用不安全的函数**： 例如，尽量避免使用strcpy、strcat等可能导致缓冲区溢出的字符串处理函数，而应使用它们的更安全版本，如strncpy、strncat等。
- **限制输入大小**： 对于从用户或其他不可靠来源获取的输入，程序员应始终验证其大小，并确保它不会超过目标缓冲区的大小。
- **使用安全的编程语言和框架**： 一些现代编程语言和框架提供了内置的缓冲区溢出防护措施，选择这些工具可以大大降低缓冲区溢出的风险。
- **用辅助工具帮助程序员查漏**，静态代码分析工具：例如，用grep来搜索源代码中容易产生漏洞的库函数（如strcpy和sprintf等）的调用；动态分析工具：使用如Valgrind这样的动态分析工具可以帮助在运行时检测潜在的内存错误。
- **用fault injection查错** 。故障注入：通过故意引入错误或异常条件（如内存损坏、网络中断等）来测试系统的容错能力。模糊测试：这是一种自动化的故障注入技术，通过向系统发送大量随机或伪随机数据来触发潜在的错误。

2 从编译器方面去防范

- **栈保护**： 编译器可以启用栈保护机制。若在程序跳转到攻击代码前能检测出程序栈已被破坏，就可避免受到严重攻击。GCC的特定选项可在生成的代码中加入一种栈保护者（stack protector）机制，用于检测对缓冲区的写入是否越界。其思想使在栈帧中任何局部缓存区与栈状态之间存储一个特殊的金丝雀（canary）值，也称为哨兵值（guard value），是在程序每次运行时随机产生的。（详见CSAPP课本第三章 `3.10.4 对抗缓冲区溢出攻击` P198~201）
  - 在函数准备阶段，在栈帧中缓冲区于所保存程序状态（如原EBP等寄存器值）之间加入一个随机生成的特定值；
  - 在函数恢复阶段，在恢复寄存器值并返回到调用函数前，先检查栈中该值是否被改变，若改变则程序异常中止；

    - 因为插入在栈帧中的特定值是随机生成的，所以攻击者很难猜测它是什么。
- **边界检查**： 编译器可以在编译时添加边界检查代码，以确保数组访问和内存操作不会超出其分配的范围。
- **警告和错误提示**： 当编译器检测到可能导致缓冲区溢出的代码模式时，它应发出警告或错误提示，以提醒程序员进行修复。

3 从操作系统方面去防范

3.1 地址空间随机化ASLR

一种比较有效的防御缓冲区溢出攻击的技术，目前在Linux、FreeBSD和Windows Vista等OS中使用。

采用ASLR，每次运行时程序的不同部分，包括程序代码、库代码、栈、全局变量和堆数据，都会被加载到内存的不同区域。这就意味着在一台机器上运行一个程序，与在其他机器上运行同样的程序，它们的地址映射大相径庭。这样才能够对抗一些形式的攻击。虽然如此，也不能提供完全的安全保障。例如，本实验的最后一个任务level 4 kaboom，利用“空操作雪橇（nop sled）”实现蛮力克服随机化攻击。

3.2 可执行代码区域限制

- 通过将程序栈区和堆区设置为 **不可执行** ， 从而使得攻击者不可能执行被植入到输入缓冲区中的攻击代码，这种技术也被成为非执行的缓冲区技术。
- 早期Unix系统只有代码段的访问属性是可执行，其他区域的访问属性是可读或可读可写。但是，近年Unix和Windows系统由于要实现更好的性能和功能，允许在栈段中动态地加入可执行代码，这是 **缓冲区溢出的根源** 。例如，“即时（just-in-time）”编译技术为解释语言（例如Java）编写的程序动态地产生代码，以提高执行性能。
- 为保持程序兼容性，不可能使所有数据段都设置成不可执行。不过，可以将动态的栈段设置为不可执行，这样，既保证程序的兼容性，又可以有效防止把代码植入栈（自动变量缓冲区）的溢出攻击。

3.3 权限管理和沙箱机制

操作系统应限制程序的权限，并使用沙箱机制将程序隔离在受限的环境中运行，以减少缓冲区溢出攻击的影响范围。

以上提到的机制是用于最小化程序缓冲区溢出攻击漏洞的常见且有效的策略。然而，不幸的是，网络安全领域始终面临着不断演变的威胁和挑战，蠕虫、病毒等恶意软件只是其中的一

部分。为了应对这些攻击，确实需要不断探索和研发更为有效的对抗技术。