C 程序中的脆弱性模式研究①

Vulnerability Patterns in C Programs

匡春光　张鲁峰　何蓉晖　(北京系统工程研究所 北京 100101)

摘　要： C 程序中脆弱性的存在严重地降低了 C 程序的安全性，为了提高 C 程序的安全性，对 C 程序中的脆弱性模式进行了研究。应用了分类、分析、归纳总结等研究方法。较系统地提出了 C 程序中存在的脆弱性模式，阐述了各种脆弱性模式的产生原因及其危害，并针对部分脆弱性模式提出了相应的避免或缓解的方法。研究结果可以用于指导 C 程序的编写，也可以用于对已有的 C 程序进行脆弱性分析，大幅度提高 C 程序的安全性，由于脆弱性种类会随着应用的发展而发生变化，今后还需要研究其它的脆弱性模式。

关键词： 脆弱性模式 缓冲区溢出 格式化字符串 竞争条件 环境欺骗

1　引言

随着社会信息化的不断深入，计算机日益成为人们工作、生活中不可缺少的工具。在计算机给人们带来方便的同时，频繁出现的与计算机相关的安全事故也给人们带来了巨大的损失。计算机安全事故的发生不是偶然的，之所以发生计算机安全事故，是因为计算机系统中存在能够引发安全事故的条件，即计算机系统中存在脆弱性[1]。对脆弱性的本质特征进行抽象和概括，就形成了脆弱性模式。深入研究 C 程序中存在的各种脆弱性模式及相应的避免或缓解方法对编写更安全的 C 程序会很有帮助。

2　脆弱性模式及避免方法

经研究发现，C 程序中存在的脆弱性主要有缓冲区溢出、格式化字符串脆弱性、竞争条件脆弱性、临时文件名可猜测和环境欺骗。下面分节阐述这些脆弱性的产生原因及危害，并针对部分脆弱性模式提出相应的避免或缓解的方法。

2.1 缓冲区溢出(buffer overflow)

缓冲区溢出是 C 程序中最普遍的安全问题，可能引发该问题的 C 标准库函数主要有 strcpy()、strcat()、gets()、sprintf()和 scanf()系列。下面以 gets()函数为例说明什么是缓冲区溢出。

C 标准库中的 gets 函数，用一个指向字符的指针(例如 s)作参数，gets 函数通过标准输入接收字符串，并把接收到的第一个字符放入 s 所指处，接下来的字符通过对 s 做加运算被依次放入 s 后的内存单元，直到遇到行结束符才停止接收数据，并在相应的内存单元放入 null 字符。如果 gets 使用的缓冲区是 n 个字节，在程序运行时，攻击者写入 n+m 个字节的数据到缓冲区中，只要这些数据中不包括行结束符，攻击者总能成功写入，即内存中这个缓冲区的后续变量很容易被覆盖。在这种情况下，有两种危险存在。首先，如果这些后续变量存储像访问控制表等安全数据，攻击者就有可能修改这些数据，从而修改访问权限。其次，如果这些后续变量存储像返回地址等数据，攻击者就有可能修改返回地址，当函数返回时就不是返回到预定的地址，而是跳转到攻击者期望的地址，从而诱使程序执行一段特定的代码。这就是缓冲区溢出。

缓解此问题的办法是不要使用函数 strcpy()、strcat()、gets()和 sprintf()，而代之以函数 strncpy()、strncat()、fgets()和 snprintf()，不要使用函数 scanf()系列。

2.2 格式化字符串(format string)脆弱性

C 程序中的格式化字符串脆弱性是由 C 的输出类标准库函数[v][f]printf()、[v]snprintf()和 syslog()引

① 收稿时间:2008-08-20

起的，这些函数都可以有格式化字符串参数，格式化字符串是其中包含特殊格式化标识的字符序列，特殊格式化标识主要包括%d、%o、%x、%c、%s、%f、%e和%n 等，它们用于指定被输出的变量的输出格式，下面以 prinf()为例说明什么是格式化字符串脆弱性。

prinf()的一种正确的使用形式为：

prinf(“str=%s”, str);

即特殊格式化标识和后面的变量参数在个数和类型上正确对应，如果 str 的值 为”abc”，以上语句将输出 str =abc，但如果程序中包括如下语句序列：

fgets(str,512,stdin);

prinf(str);

程序运行到 fgets(str,512,stdin)时，如果攻击者输入”abc %x”，其下的 prinf(str)实际上是 prinf(“abc %x”)，此语句在输出 abc 后，因为找不到和%x 对应的变量，系统将把与 str 相临的内存中的值以 16 进制的形式输出，如果攻击者输入更多特殊格式化标识，系统将继续输出内存中更多的值，这样就达到了泄露内存信息的目的。

程序运行到 fgets(str,512,stdin)时，如果攻击者输入”\x08\x08\x08\x08%12x%n”，其下的 prinf(str)实际上是 prinf(“\x08\x08\x08\ x08 %12x%n”)，此语句在输出四个 8 后，再以 12 个字节的宽度输出内存中的一个值，因为%n 的作用是把在此之前输出的字节数写到内存中，此处的内存地址由格式化字符串的头四个字符值确定，即 0x08080808，%n 之前已输出 1+1+1+1+12=16 个字节，因此系统会往内存 0x08080808 处写入 16，这样就达到了改写内存的目的。

避免此问题的办法是不要直接把用户输入的信息当作函数[v][f]printf()、[v]snprintf()和 syslog()的参数。

2.3 竞争条件(race condition)脆弱性

在某些情况下，程序的正确运行依赖于某一条件的保持，当该条件未保持足够长时间时，就会发生竞争条件脆弱性[2]。

典型的竞争条件脆弱性有 Time-of-Check-to-Time-of-Use（TOCTTOU）。下面举例说明攻击者是如何利用此问题的。

许多 C 程序经常使用一些公共可写的目录创建并打开临时文件，这些公共可写的目录一般是可以被攻击者访问的。当然，程序在通过 open(filename,O_CREAT)函数创建并打开临时文件前会通过 access(filename,F_OK)函数检查该文件是否存在，如果该文件已存在，open 操作将不执行，如果该文件不存在，open 操作将成功执行。例如，一个文本编辑器在检查文件/tmp/foo 不存在后打开文件/tmp/foo，如果在文本编辑器检查文件/tmp/foo 不存在后，但在打开文件/tmp/foo 前，一个攻击者把/tmp/foo 与/etc/passwd 连接起来，这样，被打开的真正的文件是/etc/passwd。文件打开后，攻击者可以在文本编辑器内任意输入用户名/密码，并保存输入的内容，其后果不堪设想。这就是 TOCTTOU 问题。

存在 TOCTTOU 问题的根本原因是 access 和 open 之间存在时间间隙，被访问文件的存在性在这个时间间隙内发生了变化，能发生这种变化是因为 access 和 open 都使用了参数 filename，而 filename 所指的真正文件能发生变化。

C 程序中的 TOCTTOU 问题不可避免。

2.4 临时文件名可猜测

C 的函数库中有

mkstemp(char *temp_name)、

mktemp(char *temp_name)和

tmpfile(void)

三个函数用于创建临时文件。由前两个函数生成的临时文件名是由 temp_name 和系统自动生成的六个字符组成的，由最后一个函数生成的临时文件名是由系统自动生成的。如果使用函数 mkstemp 和 mktemp，临时文件所在的目录为系统默认的临时目录，如果使用函数 tmpfile，临时文件所在的目录为 usr/include/stdio.h 定义的 P_tmpdir 来决定。很多程序员认为这样生成的临时文件名很难猜，实际情况并不是这样的，因为系统很难生成随机性很强的字符串。如果临时文件所在的目录为系统默认的临时目录，问题就更严重，因为很多用户都对系统默认的临时目录具有访问权。恶意用户可以很方便地在系统默认的临时目录下创建大量符号连接文件，这些文件的名字是程序很可能使用的临时文件名，这些文件连接的目标是重要的系统配置文件，如果创建的文件中确实存

在程序使用的临时文件，恶意用户就可以访问重要的系统配置文件，因为恶意用户可以访问程序使用的临时文件。

避免此问题的办法是不要使用函数 mkstemp 和 mktemp，使用函数 tmpfile 时，确保恶意用户对 P_tmpdir 指定的目录不具有访问权。

2.5 环境欺骗

环境欺骗通常是指 PATH 环境变量欺骗[3]。在 UNIX 系统上执行不带绝对路径的 shell 命令时，系统会到 PATH 环境变量指定的路径下去查找该命令，查找顺序也是由 PATH 环境变量决定的。假设 PATH 环境变量的内容为：/sbin:/bin:/usr/bin，如果要执行 ls 命令，系统会先到/sbin 下查找，如果找到了就执行/sbin 下的 ls，如果没有找到，则再到/bin 下查找，依此类推。如果一个程序需要执行一个外部命令，同时又只指定了命令的名称而没有给出绝对路径，那么攻击者就可以构造一个同名的命令程序，然后修改环境变量，使系统先到新构造的命令所在的目录中查找，这样，被执行的命令就被替换掉了，攻击者可以在新构造的命令程序中任意发挥。

C 的函数库中有

execl(const char *path_cmd, const char *arg, …)、

execle(const char *path_cmd, const char *arg, …, char* const envp[])、

execlp(const char *cmd, const char *arg, …) 、

execv(const char *path_cmd, char* const argv[]) 、

execve(const char *path_cmd, char* const argv[], char* const envp[])和

execvp(const char *cmd, char* const argv[])

六个函数用于执行外部命令。其中的参数 path_cmd 表示要执行的命令所在的路径及命令名，cmd 表示要执行的命令。对于函数 execlp 和 execvp，被执行的命令所在的路径是由 PATH 环境变量决定的，这些环境变量是有可能被修改的，对于函数 execl、execle、execv 和 execve，被执行的命令所在的路径可以明确指定，指定后的路径是不可能被修改的。

为了避免这个问题，当程序需要执行外部命令时，不要使用函数 execlp 和 execvp，要使用函数 execl、execle、exec 和 execve，而且要确保恶意用户对相应的目录不具有访问权。

3 结束语

本文较详细地阐述了 C 程序中存在的各种脆弱性模式，并针对部分脆弱性模式提出了相应的避免或缓解方法。通过对模式和相应的避免或缓解方法的研究，可以帮助程序员编写更安全的 C 代码，也可以帮助软件脆弱性分析者更高效地分析软件中存在的脆弱性，是一项很有实用价值的研究工作。

参考文献

1 Viega J, Bloch JT, Kohno T, McGraw G. ITS4: A Static Vulnerability Scanner for C and C++ Code . http://www.cigital.com/its4/ , 2000－03.

2 Wheeler D. Secure Programmer: Prevent Race Conditions. http://www.ibm.com/developerworks/lin-ux/library/l-sprace.html, 2004－10.

3 许治坤,王伟,郭添森,杨冀龙.网络渗透技术.北京:电子工业出版社,2005:89－95.