

C 库中易受缓冲区溢出攻击的脆弱函数分析

匡春光,王春雷,刘 强,何蓉晖

(北京系统工程研究所,北京 100101)

摘 要: 为了编写更安全的 C 程序和提高已有 C 程序的安全性,对 C 库函数中易受缓冲区溢出攻击的脆弱函数进行了分析,分析它们可能产生缓冲区溢出时的特征及如何避免缓冲区溢出. 实现了一种缓冲区溢出检测工具,能较准确地检测到 C 目的程序中的缓冲区溢出漏洞,分析结果具有实用价值.

关键词: C 库函数;缓冲区溢出;脆弱;特征;目的程序

中图分类号: TP309.2

文献标识码: A

文章编号: 1000-7180(2011)02-0189-04

Analysis of C Functions Vulnerable to Buffer Overflow

KUANG Chun-guang, WANG Chun-lei, LIU Qiang, HE Rong-hui

(Beijing Institute of System Engineering, Beijing 100101, China)

Abstract: In order to produce more secure C programs, and to improve the security of existed C programs, C library functions that are vulnerable to buffer overflow attack are analyzed. They are analyzed to obtain the features when they are used in a vulnerable way, and suggestions are given to avoid the vulnerable way. A buffer overflow detection tool that makes use of the analysis result can detect buffer overflow in C binary programs. The analysis result is practical.

Key words: C library function; buffer overflow; vulnerable; feature; binary program

1 引言

缓冲区溢出是软件系统中最常见的安全威胁^[1-2]. 缓冲区溢出攻击的形式有很多种,包括缓冲区破坏攻击、内存指针攻击、帧指针攻击等^[3],但发生缓冲区溢出的根本原因只有一个,那就是在内存拷贝时,源缓冲区所占内存空间的大小大于目的缓冲区所占内存空间的大小. 如果内存拷贝时不对目的缓冲区做边界检查,就可能发生缓冲区溢出. 由于 C 程序员可以直接对指针进行操作, C 编译器不对缓冲区做边界检查,所以 C 程序与生俱来就易受缓冲区溢出攻击^[4]. C 库函数中涉及内存拷贝的很多函数都可能产生缓冲区溢出问题.

2 脆弱函数分析

C 库中可能产生缓冲区溢出问题的函数不仅包括经常被提到的 `strcpy()`、`strcat()`、`gets()`、`scanf()`

`()`、`vscanf()`、`fscanf()`、`vfscanf()`、`sscanf()`、`vsscanf()`、`sprintf()`和 `vsprintf()`,其实还包括很多人没有意识到的 `strncpy()`、`strncat()`、`snprintf()`、`vsnprintf()`、`fgets()`、`read()`、`fread()`、`memcpy()`、`memmove()`、`bcopy()`、`memccpy()`、`memset()`、`bzero()`、`recv()`和 `recvfrom()`^[5]. 如果使用不当,这些函数很容易产生缓冲区溢出问题. 在不适当使用时,这些函数会有其独特的特征表现. 分析发现,这些函数的输入来源包括字符串、文件和网络. 下面按输入来源分节详细介绍各个函数在被不适当使用时的特征及如何避免不适当的使用.

2.1 字符串

输入来源为字符串的函数依据源数据的长度分为长度不定、长度限制不够和伪长度三类.

2.1.1 长度不定

这类函数没有比较源缓冲区和目的缓冲区的大小,但源缓冲区可能大于目的缓冲区,导致缓冲区溢

出. 这类函数包括:

`char * strcpy(char * dest, char * src)`. `strcpy` 将 `src` 所指字符串拷贝到 `dest` 所指内存空间. 其特征是 `src` 所指字符串的长度大于 `dest` 所指内存空间的大小.

`char * strcat(char * dest, char * src)`. `strcat` 将 `src` 所指字符串拷贝到 `dest` 所指字符串的尾部. 其特征是 `src` 所指字符串的长度与 `dest` 所指字符串的长度之和大于 `dest` 所指内存空间的大小.

使用这类函数时,要确保源缓冲区的大小小于或等于目的缓冲区的大小.

2.1.2 长度限制不够

这类函数在对字符串进行转换时可能会限制转换结果的长度,但如果没有把转换结果的长度限制在目的缓冲区的大小范围内,会导致缓冲区溢出. 这类函数包括:

`int sscanf(char * str, char * format, char * s, ...)`. `sscanf` 将 `str` 按 `format` 格式转换,并把转换结果保存到 `s` 等所指的内存空间中. 其特征是 `str` 所指字符串与 `s` 等对应的部分的长度大于 `s` 等所指内存空间的大小.

`int vsscanf(char * str, char * format, va_list ap)`. `vsscanf` 把 `str` 所指字符串按 `format` 格式转换,并把转换结果保存到 `ap` 中的变量所指的内存空间中. 其特征是 `str` 所指字符串与 `ap` 中的变量对应的部分的长度大于 `ap` 中的变量所指内存空间的大小.

`int sprintf(char * str, char * format, char * s, ...)`. `sprintf` 将 `s` 等按 `format` 格式转换,直到 `format` 中出现字符串结束符为止,然后把转换结果保存到 `str` 所指内存空间. 其特征是转换结果的长度大于 `str` 所指内存空间的大小.

`int vsprintf(char * str, char * format, va_list ap)`. `vsprintf` 将 `ap` 中的变量按 `format` 格式转换,直到 `format` 中出现字符串结束符为止,然后把转换结果保存到 `str` 所指内存空间. 其特征是转换结果的长度大于 `str` 所指内存空间的大小.

使用这类函数时,要把转换结果的长度限制在目的缓冲区的大小范围内.

2.1.3 伪长度

2.1.3.1 函数分析

这类函数的危险性是很多人没有意识到的. 这类函数对要写入目的缓冲区的字符串的长度进行了严格的限制. 表面上,这类函数很安全,但要写入目的缓冲区的字符串的长度有可能大于目的缓冲区的

大小,因此有可能导致缓冲区溢出. 这类函数包括:

`int snprintf(char * str, int size, char * format, char * s, ...)`. `snprintf` 将 `s` 等按 `format` 格式转换,直到 `format` 中出现字符串结束符或转换结果的长度等于 `size` 为止,然后把转换结果保存到 `str` 所指内存空间. 其特征是 `size` 大于 `str` 所指内存空间的大小.

`int vsnprintf(char * str, int size, char * format, va_list ap)`. `vsnprintf` 将 `ap` 中的变量按 `format` 格式转换,直到 `format` 中出现字符串结束符或转换结果的长度等于 `size` 为止,然后把转换结果保存到 `str` 所指内存空间. 其特征是 `size` 大于 `str` 所指内存空间的大小.

`char * strncpy(char * dest, char * src, int n)`. `strncpy` 从 `src` 所指字符串拷贝 `n` 个字符到 `dest` 所指内存空间. 其特征是 `n` 大于 `dest` 所指内存空间的大小.

`char * strncat(char * dest, char * src, int n)`. `strncat` 从 `src` 所指字符串拷贝 `n` 个字符到 `dest` 所指字符串的尾部. 其特征是 `n` 与 `dest` 所指字符串的长度之和大于 `dest` 所指内存空间的大小.

`void * memcpy(void * dest, void * src, int n)`. `memcpy` 把 `src` 所指内存内容的前 `n` 个字节拷贝到 `dest` 所指的内存空间,不管是否遇到字符串结束符,`memcpy` 都会拷贝 `n` 个字节. 其特征是 `n` 大于 `dest` 所指内存空间的大小.

`void * memmove(void * dest, void * src, int n)`. `memmove` 把 `src` 所指内存内容的前 `n` 个字节拷贝到 `dest` 所指的内存空间,不管是否遇到字符串结束符,`memmove` 都会拷贝 `n` 个字节. 其特征是 `n` 大于 `dest` 所指内存空间的大小.

`void bcopy(void * src, void * dest, int n)`. `bcopy` 把 `src` 所指内存内容的前 `n` 个字节拷贝到 `dest` 所指的内存空间,不管是否遇到字符串结束符,`bcopy` 都会拷贝 `n` 个字节. 其特征是 `n` 大于 `dest` 所指内存空间的大小.

`void * memccpy(void * dest, void * src, int c, int n)`. `memccpy` 把 `src` 所指内存内容的前 `n` 个字节拷贝到 `dest` 所指的内存空间. 不管是否遇到字符串结束符,`memccpy` 都会拷贝 `n` 个字节. 其特征是 `n` 大于 `dest` 所指内存空间的大小.

`void * memset(void * s, int c, int n)`. `memset` 将 `s` 所指内存空间的前 `n` 个字节填入 `c`. 其特征是 `n` 大于 `s` 所指内存空间的大小.

void bzero(void *s, int n). bzero 将 s 所指内存空间的前 n 个字节填入 0. 其特征是 n 大于 s 所指内存空间的大小.

使用这类函数时,要确保写入目的缓冲区的字符串的长度小于或等于目的缓冲区的大小.

2.1.3.2 实例

下面是 RealPlayer 中存在的此种类型的一处隐蔽性很强的漏洞. RealPlayer 是一款非常流行的媒体播放器. RealPlayer 播放器的墙壁时钟 (wall-clock) 功能没有正确处理 HH:mm:ss.f 时间格式, 相关代码如下:

```

925     SmilTimeValue:: parseWallClockValue (REF
(const char *) pCh)
926 {
...
957     char buf[10]; /* Flawfinder: ignore */
...
962     while (* pCh)
963     {
...
972     else if (isspace(* pCh) || * pCh == '+' || *
pCh == '-' || * pCh == 'Z')
973     {
974         // this will find the last +, - or Z... which is
what we want.
975         pTimeZone = pCh;
976     }
...
982     ++pCh;
983     }
...
1101     if (pTimePos)
1102     {
1103         //HH:MM...
....
1133     if (* (pos-1) == ':')
1134     {
....
1148     if (* (pos-1) == '.')
1149     {
1150         // find end.
1151         UINT32 len = 0;
1152         if (pTimeZone)
1153         {
1154             len = pTimeZone - pos;
1155         }
1156     else

```

```

1157     {
1158         len = end - pos;
1159     }
1160     strncpy(buf, pos, len); /* Flawfinder: ignore
*/

```

1160 行的 strncpy 从 pos 所指字符串拷贝 len 个字符到 buf 所指内存空间. 1154 行和 1158 行计算 len, 可见 len 可能和 pTimeZone 相关, 由 975 行知, pTimeZone 和 pCh 相关, pCh 是外部输入, 它的长度是没有限制的, 这就意味着 len 的长度实际上是没有限制的, 而由 957 行知, buf 的长度为 10. 因此在 1160 行可能出现缓冲区溢出.

2.2 文件

输入来源为文件的函数依据源数据的长度分为长度无限制、长度限制不够和伪长度三类.

2.2.1 长度无限制

这类函数从文件读入字符, 对字符数无限制, 因此, 读入的字符数可能大于目的缓冲区的大小, 导致缓冲区溢出. 这类函数包括:

char * gets(char * s). gets 从标准输入读入字符并存到 s 所指的内存空间, 直到出现换行字符或读到文件尾为止. 其特征是从标准输入读入的字符串长度大于 s 所指内存空间的大小.

不应该使用这类函数.

2.2.2 长度限制不够

这类函数先从文件中读入字符串, 然后对读入的字符串进行转换, 在转换时可能会限制转换结果的长度, 但如果没有把转换结果的长度限制在目的缓冲区的大小范围内, 会导致缓冲区溢出. 这类函数包括:

int fscanf(FILE * stream, char * format, char * s, ...). fscanf 把从 stream 中读入的字符串按 format 格式转换, 并把转换结果保存到 s 等所指的内存空间中. 其特征是从 stream 中读入的字符串与 s 等对应的部分的长度大于 s 等所指内存空间的大小.

int scanf(char * format, char * s, ...). scanf 把从标准输入读入的字符串按 format 格式转换, 并把转换结果保存到 s 等所指的内存空间中. 其特征是从标准输入读入的字符串与 s 等对应的部分的长度大于 s 等所指内存空间的大小.

int vscanf(char * format, va_list ap). Vscanf 把从标准输入接收的字符串按 format 格式转换, 并把转换结果保存到 ap 中的变量所指的内存空间中.

其特征是从标准输入接收的字符串与 ap 中的变量对应的部分的长度大于 ap 中的变量所指内存空间的大小。

int vfscanf(FILE * stream, char * format, va_list ap). vfscanf 把从 stream 接收的字符串按 format 格式转换, 并把转换结果保存到 ap 中的变量所指的内存空间中. 其特征是从 stream 接收的字符串与 ap 中的变量对应的部分的长度大于 ap 中的变量所指内存空间的大小。

使用这类函数时, 要把转换结果的长度限制在目的缓冲区的大小范围内。

2.2.3 伪长度

这类函数的危险性也是很多人没有意识到的. 这类函数对从文件中读入的字符串的长度进行了严格的限制. 表面上, 这类函数很安全, 但从文件中读入的字符串的长度有可能大于目的缓冲区的大小, 因此有可能导致缓冲区溢出. 这类函数包括:

char * fgets(char * s, int size, FILE * stream). fgets 从 stream 所指的文件内读入字符并存到 s 所指内存空间, 直到出现换行符、读到文件尾或已读了 size-1 个字符为止. 其特征是 size 大于 s 所指内存空间的大小。

int read(int fd, void * buf, int count). read 从 fd 所指文件读取 count 个字节, 并保存到 buf 所指内存空间. 其特征是 count 大于 buf 所指内存空间的大小。

int fread(void * ptr, int size, int nmemb, FILE * stream). fread 从 stream 中读取 size * nmemb 字节的数据并存到 ptr 所指内存空间. 其特征是 size * nmemb 大于 ptr 所指内存空间的大小。

使用这类函数时, 要确保从文件中读入的字符串的长度小于或等于目的缓冲区的大小。

2.3 网络

输入来源为网络的函数只有伪长度一类。

这类函数的危险性同样是很多人没有意识到的. 这类函数对从网络读入的字符串的长度进行了严格的限制. 表面上, 这类函数很安全, 但从网络读入的字符串的长度有可能大于目的缓冲区的大小, 因此有可能导致缓冲区溢出. 这类函数包括:

int recv(int s, void * buf, int len, int flags). recv 接收从远端主机指定的 socket s 传来的 len 个字节大小的数据, 并把这些数据保存到 buf 所指的

内存空间. 其特征是 len 大于 buf 所指内存空间的大小。

int recvfrom(int s, void * buf, int len, int flags, SOCKADDR * from, int * fromlen). recvfrom 接收从远端主机 from 指定的 socket s 传来的 len 个字节大小的数据, 并把这些数据保存到 buf 所指的内存空间. 其特征是 len 大于 buf 所指内存空间的大小。

使用这类函数时, 要确保从网络读入的字符串的长度小于或等于目的缓冲区的大小。

3 结束语

文中分析了 C 库函数中可能产生缓冲区溢出问题的函数, 分析它们可能产生缓冲区溢出时的特征及如何避免缓冲区溢出. 分析结果对编写安全的 C 程序有参考价值, 对检测 C 程序中的缓冲区溢出漏洞有指导作用。

参考文献:

- [1] Olatunji Ruwase, Monica S Lam. A practical dynamic buffer overflow detector[C]// Proceedings of the Network and Distributed System Buffer Overflow Symposium. Santa Clara, 2004:159-169.
- [2] 郑宏超, 范隆, 岳素格, 等. 基于库替换技术的单粒子效应故障注入仿真[J]. 微电子学与计算机, 2009, 26(7):59-61.
- [3] Manish Prasad, Tzicker Chiueh. A binary rewriting defense against stack based buffer overflow attacks[C]// Proceedings of USENIX'03 Annual Technical Conference. San Antonio, Texas, USENIX Press, 2003:211-224.
- [4] 付剑晶, 黄金钟. 一种缓冲区溢出攻击的编译数据空间对策[J]. 微电子学与计算机, 2008, 25(6):190-193.
- [5] 徐千洋. Linux C 函数库参考手册[M]. 北京: 中国青年出版社, 2002.

作者简介:

- 匡春光 女, (1971-), 副研究员. 研究方向为计算机安全.
王春雷 男, (1977-), 助理研究员. 研究方向为计算机安全.
刘 强 男, (1979-), 助理研究员. 研究方向为计算机安全.
何蓉晖 女, (1977-), 助理研究员. 研究方向为计算机安全.