C 语言的内存漏洞分析与研究

刘素娇

（河南护理职业学院，河南 安阳 455000）

摘　要：C 语言允许开发人员直接对内存操作，方便灵活，提高了程序的执行效率，但这也在一定程度上牺牲了安全性。程序一旦发生内存错误，将会非常棘手，编译时不能自动发现这些漏洞，只有在程序运行时才会出现，而这些漏洞所导致的后果（如数据错误、内存泄露、系统崩溃、系统被攻击、用户数据被窃取等）很难捕捉到，测试也很难重现。

关键词：内存错误；指针；内存测试

DOI:10.16184/j.cnki.comprg.2019.06.057

1　引言

随着并行技术、分布式技术以及虚拟技术的相互融合、相互促进，云计算、大数据和人工智能有了突飞猛进的发展，软件系统规模越来越大，结构越来越复杂，软件系统稳定性和可靠性变得越来越难保障。为了保障软件系统的稳定性和可靠性，各软件公司在会在软件生命周期各个阶段投入大量人力和物力来保证软件质量。

C 语言具有数据类型丰富多样，开发时灵活方便：指针直接对硬件操作即对物理地址的直接访问、具有大量库函数，这些使得程序执行效率高；并且具有兼顾低级编程语言和高级编程语言、通过对位、字节和内存地址进行操作进而对系统软件进行访问和操作等优势。由于 C 语言对内存空间的访问操作、接近底层硬件，C 语言程序具有执行效率高、可移植性强等特点受到操作系统、企业数据库管理系统、嵌入式软件等相关软件的青睐。

但 C 语言缺乏对内存边界检查，内存越界主要包含数据越界读写（如 C 语言编译器没有对数组或缓冲区进行边界检查）、“野指针操作”和堆栈溢出等，而这些都有可能程序存在严重的安全漏洞，在一定程度上降低了程序的可靠性和安全性；不对接收参数类型做校验，这都使得程序存在一定的安全隐患；安全保护机制和异常处理机制的缺失使得 C 语言在编译时不能及时发现漏洞或异常，但在程序运行后有可能会出现各种安全隐患，而这些安全隐患或漏洞一旦被攻击者或非法用户窃取信息或恶意篡改，有可能造成无法估计的损失，程序的安全性大大降低。

2　常见内存错误分析

计算机的内存采用的是分区管理，程序与程序所占用的内存是相互独立的，是不能相互访问的，并且每个程序的内存采用的也是分区管理。通常情形下，C 语言程序的内存分配分为以下几种方式：

（1）从静态存储区域分配。生命周期较长：程序所占用内存在编译时已分配好，变量值所占用内存从函数使用开始到程序的整个运行期间都存在。如全局变量、static 变量。

（2）在栈上创建。生命周期只存在于函数的执行过程中，分配容量有限。在函数执行时，函数内局部变量的存储单元在栈上创建，随着函数执行结束存储空间被自动释放。

（3）从堆上分配，亦称动态内存分配。内存空间的生存周期由开发人员调用 malloc 或 new 申请内存时开始，到调用 free 或 delete 时被释放时结束。如果调用了 malloc 或 new，却没有调用 free 或 delete 来释放存储空间，容易发生内存泄露。

总体来说，在 C 语言中常见的内存漏洞是缓冲区溢出、指针非法使用和内存泄露。

2.1　缓冲区溢出

缓冲区溢出的原因主要是因为在 C 语言中缺失对数组下标越界检查或缓冲区边界检查，是指向已经分配好固定存储空间存放数据，如果向内存空间写入数据时超出了已分配好的空间，就会占用相邻的存储空间，覆盖相邻内存单元的内容，从而引发不可预知的程序错误。

作者简介：刘素娇（1982–），女，实验师，硕士，研究方向：计算机基础教学及网络。

收稿日期：2019–03–21

（1）数组下标越界主要是因为对于数组来说为其分配的存储空间是连续的，由于开发人员的疏忽经常会导致数组下标 “多 1”或“少 1”，尤其是在 for 循环中，循环次数经常会多 1，此时容易发生操作越界，发生不可预知的错误。下面的程序段就是因为对数组下标操作错误造成的。

```
如：int a[10]={ 0,1,2,3,4,5,6,7,8,9 };
    for(i=0;i<=10;i++)
      a[i]=a[i]+3;
```

程序的第一行语句是定义数组，第 2、3 行是 for 循环语句，数组 a [10] 只有 10 个元素，而 for 循环语句中的 “i<=10”却使得 for 循环执行了 11 次，在最后一次循环执行过程中语句 “a [10] =a [10] +3；”改变了存储数组区域相邻存储单元的内容，发生了缓冲区溢出，造成不可预知的程序错误。

（2）另外对于字符串的结束判断只是通过识别 ‘\0’来进行识别，此种情况下也很容易发生缓冲区溢出。

（3）在 C 语言的标准库或库函数调用中，对数据类型的不进行强制性检查造成越界操作，从而发生溢出[1]。

2.2 指针的非法使用

允许开发人员直接对内存进行操作，极大地提高了程序的运行效率。但是高效率也意味着高风险。开发人员在利用指针对内存操作时经常会发生指针操作错误从而引起程序出错甚至崩溃。常见的指针非法使用通常包含以下几种：

（1）指针变量没有被初始化，或者是未分配空间的指针

在程序运行前指针的指向不确定，此时如果在没有对其进行初始化的情况下，就开始引用，容易出现引用错误或是指向毫无意义的内容，从而引起系统崩溃或错误。如：

```
char *p；*p=´a´;
```

这是因为指针变量在创建后不会自动为 NULL，它的缺省值是随机的，指向的内容是不确定的，所以在创建指针时应先初始化，将其置为 NULL 或者是指向合法的内存。

（2）使用未初始化指针的内容

此类指针的使用错误主要是因为指针指向的内容没有被初始化，而进行相应的操作造成的。如：char *p, m； p=new char; m=*p。

（3）使用已释放的指针

在释放动态申请的内存空间时，采用的 free 或 delete 释放掉的只是指针指向的内存空间，而相应的指针的地址没有改变，指向的是内存垃圾，指针此时成为所谓的 “野指针”。如果释放空间后，没有把相应的指向内存空间的指针置 “NULL”，会让开发人员误以为是合法指针。并且此时的 if （p ! = NULL） 起不到防错作用，因为即使 p 不是 NULL 指针，指向的内容也是不合法的内存空间。

2.3 内存泄露

不管是经验丰富或是新入门的开发人员，开发出的程序都会不同程度地存在这个问题。即便是 Windows 或 Linux 也存在着内存泄露。内存泄露主要因为在函数体内利用 malloc 或 realloc 或 new 动态申请内存，随着函数的调用结束，开发人员很容易忘记利用 free 或 delete 来释放这块内存空间，此种情况下内存空间在系统中将处于被占用的状态导致其他函数无法使用，随着程序的不断运行，可用的内存空间越来越少，程序运行也越来越慢，直至系统死机，即发生了所谓的内存泄露。

在动态申请分配内存空间时，申请与释放必须成对出现，即 malloc 与 free 的使用次数以及 new 与 delete 的使用次数一定要相同，否则将会发生内存泄露。如：

```
for(i=1;i<=10;i++)
{
*p=(double*)malloc(sizeof(double));
*p=sqrt(i);
}
free(p);
```

在此 for 循环语句中，动态申请了 10 次内存，但是 free 函数在上面的语句中只进行了一次释放内存，而前 9 次申请的内存空间没有被释放，造成内存资源泄露。

在动态申请释放内存空间时，也一定要注意释放内存空间的顺序，即对于多级连续内存空间的分配先申请大空间再申请小空间，而释放则是先释放小空间再释放大空间[2]，若先释放大空间的话，小空间的内容就会造成内存泄露。

3 内存安全的检测技术

C 语言的内存安全错误检测技术有静态检测和动态检测。

(下转第 160 页)

源。除此之外，医院应建立基层签约服务制度，对于不同人群进行差异化服务，满足不同患者对于医疗服务的不同需求。分级诊疗还应健全医疗服务的价格形成机制。通过大数据分析，合理制定医疗服务价格，建立动态的医疗服务价格机制，形成良性的市场竞争机制。面对分级诊疗的快速推进，医院应明确各个部分的职责，及时制定相关政策，开展医护人员和行政管理人员的培训，制定定期考核机制，共同推进分级医疗服务体系的建设，实现医疗资源的合理配置。

5 结语

“互联网+”时代智慧医院的建设，将推动我国医疗服务的发展，将趋向于智能化、高效化、信息化，这不仅是社会主义市场经济发展下的产物，也是人民群众对于医疗服务体系的最高要求。互联网信息技术在医疗行业的应用，推动了我国医疗健康与科学技术的全方位融合，有利于改善地区间医疗资源配置的差异，提高我国医疗资源的利用率，使我国医疗行业将呈现出蓬勃发展的新局面。

参考文献

[1] 秦荣昌. 智慧医院建设探讨与研究. 智能建筑, 2016.

[2] 陈秋晓. 智慧医院建设存在的问题与建议. 医院管理论坛, 2013.

(上接第 153 页)

3.1 静态检测

静态检测是指代码处于非运行状态下对程序源代码或二进制代码进行检测，以其找到发生程序错误的代码片段。静态检测的过程可以分为创建源程序代码、建立漏洞特征库、判定漏洞以及检测结果分析等 4 个步骤。常用的静态检测技术有数据流分析和符号执行。静态检测技术由于不运行代码，执行效率高但也存在着路径覆盖不全面和建模不完全等缺点，有可能会存在错报或漏报的问题[3]。

3.2 动态检测

动态检测技术主要是利用插桩技术在代码运行时进行代码检测。主要的内存错误检测技术有基于对象技术、值验证技术、影子内存技术、扩展指针技术和基于指针技术。此 5 种技术都在不同程度上满足对内存错误进行了检测的目的[4]。

内存漏洞的动态测试当前面临漏报率非常高、依赖测试用例、分支覆盖率低、影响效率等问题和困难。所以测试的结果也非常依赖测试人员的能力和测试用例的设计。这些问题都给软件动态测试精度带来困难。因为静态检测和动态检测都存在不同的优缺点，一般是静态和动态结合使用。

4 结语

C 语言允许开发人员对内存操作，简单方便灵活，受到开发人员尤其是系统开发人员的欢迎。但程序一旦发生内存错误，将会发生数据错误、系统运行缓慢、系统崩溃、系统被攻击甚至系统用户敏感信息被篡改或窃取等不可预知的错误，并且漏洞出现后错误又不好定位，测试很难重现，尤其是内存泄露，在编译时不会发现错误，程序运行后，也不会立即崩溃，只有运行一段时间之后，程序会出现运行缓慢直至死机或崩溃。

首先分析了 C 语言中常见的内存错误，并对产生错误的原因进行了分类，又对检测内存错误的关键技术进行了浅析。下一步的研究重点是对检测内存的关键技术进行更深入的研究。

参考文献

[1] 陈小全，薛锐. 程序漏洞：原因、利用与缓解——以 C 和 C++语言为例 [J] . 信息安全学报，2017,(2) : 42–48.

[2] 吕维梅，刘坚. C/C++程序安全漏洞的分类与分析 [J] . 计算机工程与应用，2005: 124–125.

[3] 田雪. C 程序静态分析中并发性漏洞检测技术的研究 [D] . 北京：北京邮电大学，2018.

[4] 严俊琦. C 程序内存安全错误的运行时检测技术研究与实现 [D] . 南京：南京航空航天大学，2017.