第30卷 第2期 计 算 机 工 程 2004年1月
Vol.30 №2 Computer Engineering January 2004
· 博士论文 · 文章编号：1000—3428(2004)02—0028—03 文献标识码：A 中图分类号：TP311.52

C源代码静态安全检查技术

刘海燕，杨洪路，王 崛
(装甲兵工程学院信息工程系，北京100072)

摘 要：对源代码进行安全检查就是在程序运行之前通过分析源程序发现潜在的安全缺陷。该文分析了C语言源程序中可能发现的安全问题，介绍了目前静态代码安全检查的技术和方法以及面临的困难，最后，给出了一些提高程序安全性的建议。

关键词：安全审查；安全缺陷；C源代码；静态代码分析

Static Security Examining for C Source Code

LIU Haiyan, YANG Honglu, WANG Jue
(Department of Information Engineering, Armored Force Engineering Institute, Beijing 100072)

【**Abstract**】A static security examiner is to find vulnerabilities by statically analyzing source codes before they are executed. The paper analyzes what possible problems can be found in C source codes, introduces some current researches on static code security examining as well as those difficulties encountered. As a conclusion, it gives some advice for improving program security.

【**Key words**】Security examining；Security vulnerability；C source code；Static code analysis

信息社会里，互联网迅速壮大，依托于网络的业务蓬勃发展，人们不得不面临网络安全问题。据权威机构统计，目前因特网上20%的企业的防火墙、80%的计算机网络包含严重的安全缺陷。研究表明，所有的入侵都是因为软件本身存在漏洞。当前的信息处理和通信系统越来越复杂，难免会存在这样或那样的一些设计者本人在设计时并没有意识到的在设计、实现以及管理中存在的缺陷，这些漏洞常被他人利用来绕过安全策略。对源程序进行安全审查的目的就是在程序设计阶段，在系统运行之前通过分析源代码发现潜在的安全缺陷。

本文讨论C语言源代码的静态安全检查。C语言是一个开放的语言，语法形式自由，不同的人可以编写出实现同一功能而源代码形式多样的程序。使用C能编写高性能程序，包括系统程序和应用程序，但是，C语言与它的支持库本质上是不安全的，无意中的失误可能导致非常严重的安全漏洞,尤其是网络程序。

1 C语言程序的安全问题

(1)内存访问错

C语言程序中引发的安全问题绝大多数属于内存访问错误，内存访问错误可能由数组、指针或内存管理造成。对指针和数组缺乏边界检查是造成内存访问错误的根源。据统计,超过50%的软件故障是由指针和数组访问造成的[1]。被列为攻击手段之首的“缓冲区溢出攻击”就是利用此类故障。

首先，仔细分析一下C语言程序能够使用的内存是怎样分配的：

·从静态存储区域分配，这块内存在程序的整个运行期间都存在，例如全局变量、static变量。

·在栈上创建，函数内的局部变量的存储单元都可以在栈上创建。

·从堆上分配，亦称动态内存分配。程序在运行的时候用malloc或new申请任意多少的内存，程序员自己负责在何时用free或delete释放内存。动态内存的生存期由程序决定，使用非常灵活，但问题也最多。

针对内存从分配、使用到释放的整个过程，可能发生如下内存错：

1)内存分配未成功，却使用了它。典型的情况就是从堆中动态申请内存，许多程序总假设内存分配成功，继续其后续操作，但实际上，内存分配可能不成功，程序必须在使用内存之前检查指针是否为NULL以进行防错处理。

2)内存分配虽然成功，但是尚未初始化就引用它。在C语言中，内存的缺省初值没有统一的标准，尽管有些时候为零值。如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未被初始化的变量，可能会导致程序错误。

3)内存分配成功并且已经初始化，但操作越过了内存的边界。这类错误在C语言中又称为空间访问故障，即通过指针或数组访问合法范围外的空间。C语言缺乏数组边界检查，数组下标越界是一种典型的空间访问故障。如在使用数组时经常发生下标“多1”或者“少1”的操作，特别是在for循环语句中，循环次数很容易搞错，导致数组操作越界。

4)忘记了释放内存，造成内存泄露。内存泄漏是黑客常使用的一种方法，其根本原因是某个错误的函数，忘记释放动态申请的内存空间，导致每被调用一次就丢失一块内存。刚开始时系统的内存充足，你看不到错误，因而调试阶段不能发现错误，当终于一天内存耗尽程序突然死掉，系统将提示：内存耗尽。

5)释放了内存却继续使用它。这类故障也称为时间访问故障，即在其生存期之外访问内存空间，一个典型的时间访问故障是访问一个已经释放的堆分配单元，由于释放的单元归还系统后可以重新分配，因此再次访问时具有不确定的值。

内存访问错误很难检测和修改，因为：

·编译器一般不能自动发现这些错误，通常是在程序运行时才能捕捉到。

·内存访问错误一般没有明显症状，除非在异常的情况下。

·导致内存访问错误的异常条件可能不能再现，增加了差错纠错的困难；

·一旦错误再现，也可能很难把程序错误与内存访问错误联系。

基金项目：“十五”预研项目(413290205)
作者简介：刘海燕(1970－)，女，博士、副教授，主研方向为网络安全；杨洪路，硕士生；王 崛，硕士
收稿日期：2003-01-02 E-mail：lhy940@163.com

(2) 缓冲区溢出

1988年大规模蠕虫传染了成千上万台联网主机，罪魁祸首就是缓冲区溢出漏洞。在过去10年里，尽管已经意识到缓冲区溢出的危害，但缓冲区溢出攻击仍在增长，据CERT/CC统计，在过去10年里，在所有的脆弱性中缓冲区溢出漏洞占据了50%，而且这个数据随时间的流逝还在增长，作为一类内存访问错误，由于它的重要性，因此在此进一步分析缓冲区溢出的根源。

缓冲区溢出就是越过了数组边界进行读写。标准C库提供的许多字符串操作，如gets、strcpy、strcat、sprinf，它们本质上是不安全。程序员有责任检查这些操作能否溢出缓冲区。

例如：程序执行gets(str)指令，str是内存一个连续的存储区的首地址，gets从程序的标准输入读取文本，把第一个字符放在这个存储区的首地址，后续的字符依次存放，直到从标准输入读到新行或文件结束符，在对应的存储区放上空字符/0。程序员没法指出一个缓冲区有多大。因此，如果读入的文本长度超过给定的存储区长度，那么，读入的字符将覆盖str所代表的存储区的后续空间。这就是缓冲区溢出。

缓冲区溢出可能造成两个风险：1)与该缓冲区相邻的内存空间很容易被覆盖，如果该空间存储的信息非常关键，那么可能造成非常严重的安全隐患；2)通过覆盖运行栈中函数返回地址，攻击者可以引诱程序执行任意代码，这类栈溢出是目前最常见的安全缺陷。

在C库函数中，每个这样的字符(函数都有相应的安全函数对应：strncpy，strncat，snprintf等，在使用这些函数时，可以由程序员指出实际缓冲区大小，避免发生缓冲区溢出。

例如，strncpy(dst,src,10)指明，从src向dst拷贝不多于10个字符。

然而，这些替代函数也有不足：一方面用起来不太方便，有些系统不支持这些函数；另一方面，跟踪缓冲区实际大小的任务仍由程序员来完成。

(3) 竞争条件(race condition)

所有操作系统都有一些陷入(trapdoor)，允许特权用户完全控制系统，竞争条件是在多进程环境里，特权程序或与特权程序并发运行的程序中可能被利用的一类缺点。

例如：一个超级用户为一个普通用户创建一个文件(Createfile)，并把该文件的所有者更改为该普通用户(chown)，但在更改所有权之前，该用户把此文件删除，并建立了此文件到系统关键文件，如/etc/password的链接，这样，所有权的更改致使系统文件被这个普通用户拥有。这个例子说明，在多进程的环境里，创建(create)和更改(change)必须是原子，其完整性不可破坏。

进程访问文件的方式有两种：一种即通过完整的文件路径访问文件；另一种是通过文件描述符访问文件。在系统内部，文件系统在概念上讲是一棵树，内部结点代表目录，叶结点代表文件，路径名指出了从树根到要访问的文件或目录所经过的各结点。每个结点记录了它的下层结点的地址，因此通过路径访问文件类似于间接指针访问。文件描述符是在每个进程的基础上，为每个文件赋予的一个标志，它直接与目标绑定，当访问文件时，直接通过文件描述符，而不需要通过文件系统一步一步查找，因此它的使用方式类似于直接指针。

现假设有两个系统调用要访问文件，如果两次访问都使用文件描述符，他们本质上是原子的。因为文件描述符是以进程为基础的，其它进程不能改变文件描述符与目标的绑定。但如果一个访问以路径形式进行，那么其它的进程可以更改路径与具体目标文件的绑定，这就可能发生竞争条件。

具体地,如果相邻的两个系统调用都使用路径访问同一个文件，那么可能存在竞争条件，如果一个使用路径，一个使用文件描述符，那么，如果第一个系统调用是把文件描述符与路径相关，则不会发生竞争条件，否则可能发生竞争条件。如果两个系统调用都使用文件描述符，那么这两个系统调用间不会发生竞争条件。

(4) 随机数的使用

C程序中涉及许多随机数的选取，例如，创建一个临时文件，用于存取一些临时数据，创建一个挑战号、创建密钥，出于安全性考虑，临时文件名和挑战号等使用由系统函数rand产生的随机数，以防攻击者猜测。但是，系统提供的rand是一个伪随机数，其内部的实现根据给定的种子能够产生可重复的输出值。因此，需要程序员选定一个安全的种子。如果不精心选择安全的种子，对精明的攻击者来说，使用随机数同使用固定数一样可以猜测。

(5) 异常控制

C语言没有异常处理机制，所有的异常检测和处理都必须由程序员主动完成，如果程序员没有明确处理异常情况，那么异常发生时系统将出现不可预测的错误。例如：

```
FILE *fp;
char buffer[128]
fp=fopen(“/etc/password”,“r”);
fgets(buffer,sizeof(buffer),fp);
```

这个程序假设系统一定有“/etc/password”这个系统文件，并且打开总能成功，如果这个文件被恶意地删除，或者程序运行的权限不够，或者其他程序正在独占使用该文件，致使打开不成功，那么，后续的读操作将发生核心错(core dump)。

因此C程序员有义务增加异常检测和处理代码,进行主动防错设计。从C发展而来的Java语言提供了非常强大的异常处理机制，强制程序员对“可检测异常”进行处理。

(6) 空指针引用

空指针即没有指向任何合法存储空间的指针，例如：

```
typedef char *FileContent;
Filecontent file1;
Filecontent file2;
…
strcpy(file2,file1);
```

该程序本意是让file1指向源，让file2指向目标，通过strcpy(file2, file1)把file1指向的内容拷贝到file2中。一般情况下，这种拷贝不会有问题，但是当file1没有指向任何合法的静态、局部或动态分配的空间时，程序运行出错。为此程序在使用指针时，不仅要关心指针本身是否定义，还要关心指针是否已经指向了合法的内存空间。应该检查从file1定义到其引用的每条可能路径，检查file1是否一定指向了合法的内存空间。

2 C语言静态安全审查技术

C语言使程序员很容易编写出不安全的程序，因此提供一个自动工具来帮助检测这类编译阶段发现不了、运行阶段又很难定位的安全相关的编程错误非常有必要。

这种静态代码分析工具分为两类：

1)在源程序编写完成、编译检查正确、初步测试运行正确之

后，把整个源文件作为分析目标，由分析工具完整细致地分析其潜在的安全缺陷；

2)在编写程序过程中，递进地进行安全检查，ITS4就属于后者，它嵌入语言的编辑环境，如Emac和Visual C++，在编程的过程中，以特殊的颜色提示可能的安全缺陷。

无论属于哪一类，静态代码分析工具的基本工作原理都是一样的：从前向后逐行读入源程序代码，定位可能的嫌疑，逐步深入分析，报告分析结果。例如要发现程序中由字符串操作引入的缓冲区溢出缺陷：

首先寻找strcpy、sprintf、gets、strcat等的所有出现，然后分析这些函数的参数，对strcpy(dst,"\N")这样的源为固定串的，一般不会发生缓冲区溢出，对str(dst,src)这样的源为一个变量的，要进一步分析，如果src是程序内部计算的结果，一般也不会发生缓冲区溢出，如果src的长度与用户或网络输入直接相关，则它是潜在的缓冲区溢出点。

目前国外已经出现了关于C和C++代码的静态安全检查工具，它们一般致力于某类缺陷的发现，还不太实用。

1)ITS4：由J. Viega等人研制的ITST4是目前检测最全面的工具[2]。它不像真正的编译器一样分析程序设计语言的语法，它只是把输出文本划分为一个个片断，再把每个片断与它自己的一个"嫌疑数据库"进行比较，如果属于嫌疑，则进一步实行启发式判断。1.0.2版本的ITS4包含131条嫌疑记录，其中大部分涉及关于文件访问的竞争条件，还有相当一部分是有关缓冲区溢出的，此外还有伪随机数等，ITS4主张，安全检查应该在可靠性和可用性方面达成平衡，也就是说，为了提高检查效率，宁可失去部分可靠性。

2)M. Bishop等人研制的文件访问竞争条件检测技术[3]：该技术检查程序中每两条相邻的关于文件访问的语句，发现其中可能存在的竞争条件。该技术不能处理当两条相邻的文件访问语句处于不同的函数中，以及使用变量作为文件名的情形。

3)D. Wagner 等人专门研究了缓冲区溢出缺陷的自动检测技术[1]，它把缓冲区溢出的检测形式化为整数范围的约束问题，通过整数约束问题求解，判断潜在的缓冲区溢出缺陷。

其它相关研究有：研制C语言的安全子集作为安全的程序设计语言、研制安全编译器，对一般的指针数组操作进行安全检查、把静态代码审查与运行栈监测等动态技术结合等[4]。我国"十五"期间也开始了关于静态代码审查技术的研究，目前尚未见到相应的技术和成果。

3 C语言静态安全检查技术面临的困难

静态分析工具对提高C和C++编写的软件的安全性有重要作用，但目前看来，这样的工具有许多技术难题：

1) C语言自由的风格致使其不太适合静态分析，C语言大量地使用指针，而指针具有动态性和别名特点，很难进行静态跟踪。

2) 在多进程/多线程环境下进行静态分析很困难。在实际的应用程序中，线程越来越普遍，线程之间的数据共享是安全的一个重要因素。

3) C语言缺乏异常处理机制的支持，使得程序设计时只关注系统功能的实现，较少考虑异常情况的检查与处理。

4 结论和建议

C语言是功能强大而风格自由的程序设计语言。静态分析工具能够在程序运行之前，通过检查源程序发现潜在的安全漏洞，对提高程序的安全性具有重要意义。但是，由于C语言本身的性质，许多问题很难在静态分析过程中确定，因此，静态检查工具不可能是万能的。我们一方面强烈建议程序员时刻保持高质量程序设计的观念，进行主动防错设计，在这个方面，可以参考林锐博士撰写的《高质量C++/C编程指南》。另一方面，对那些比较关键的系统，提醒使用多种安全手段相结合的安全防护体系，例如，动态资源监控、栈监视，以及漏洞扫描、入侵检测等，确保系统安全。

参考文献

1 Wagner D,Foster J,Brewer E,et al. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities.In Proceedings of the Year 2000 Network and Distributed System Security Symposium(NDSS), San Diego, CA,2000:3-17

2 Biega J.ITS4:A Static Vulnerability Scanner for C and C++ Code. http: //www.rstcorp.com

3 Bishop M,Dlger M.Checking for Race Conditions in File Access. Computing Systems,1996,9(2):131-152

4 Austin T M,Breach S E,Sohi G S.Efficient Detection of All Pointer and Array Access Errors. http://citeseer.nj.nec.com/1096.html

☆☆☆

(上接第4页)

挖掘时，用户创建的服务之间有信息的交互。当某个服务的状态改变时，为保证系统能正常运转，应该及时地发送改变信息给其他服务。OGSA模型提供了通知机制来解决这个问题。实现的方式是定义通用的抽象形式和服务接口。

(5) 管理能力。用户可能会同时创建很多挖掘服务，用于不同的目的。为达到系统的友好性，必须随时可以监控和管理这些实例。OGSA模型通过管理接口，提供管理能力解决这个问题。

5 结论与未来的工作

在本文中给出了一个生物数据网格框架，描述了在调控基因的生物数据网格上进行数据挖掘的过程和OGSA提供的一些底层机制作为数据挖掘过程的实现基础。随着生物科学的发展，适用于各种特定任务的生物数据网格将会成为网格应用研究的一个热点。网格平台软件存在着统一的趋势，未来的工作可以在OGSA的基础上，集中研究解决在网格环境下实现不同数据源的数据提取、整合与动态更新和开发用于生物数据网格的核心网格软件。

参考文献

1 都志辉,陈 渝,刘 鹏等.网格计算.http://grid.cs.tsinghua.edu.cn/, 2003-04

2 Chervenak A, Foster I,Kesselman C,et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications, 2001,23:187-200

3 Foster I, Kesselman C, Nick M J,et al. Grid Services for Distributed System Integration. Computer, 2002 ,35(6):37-46

4 Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid:Enabling Scalable Virtual Organizations.Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid,2001:6-7

5 Stockinger H,Samar A, Allcock B,et al.File and Object Replication in Data Grids.Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing ,2001:76 -86

6 Laszewski G V,Russell M,Foster I,et al.Community Software Development with the Astrophysics Simulation Collaboratory.Concurrency and Computation:Practice and Experience,Concurrency Computat.: Pract. Exper, 2002, 14:12891301 (DOI: 10.1002/cpe.688)