CoSy C语言编译器程序缓冲区溢出研究

史 岩,李蜀瑜,丘 征,陈长胜
(陕西师范大学 计算机科学学院,陕西 西安 710062)

摘 要:CoSy是ACE公司开发的编译器构架,为了保证CoSy C编译器输入程序的安全性,避免产生编译器缓冲区溢出问题,提出了编译器缓冲区溢出判断模型。根据C源程序编译器缓冲区溢出漏洞的特征,建立了编译器缓冲区溢出判断模型;给出了重建CoSy中间表示CCMIR(Common CoSy Medium-level Intermediate Representation)的方法;最后,给出了CCMIR程序安全性判定算法。实验结果表明,这种模型可以有效地判断输入程序的安全性。因此,通过重建CCMIR模型可以有效地避免CoSy C语言编译器编译过程中的缓冲区溢出问题。

关键词:中间表示;CoSy;缓冲区溢出;XML
中图分类号:TP314 **文献标识码**:A **文章编号**:1673-629X(2012)06-0093-04

Research on Procedure Buffer Overflow of a CoSy C Compiler

SHI Yan, LI Shu-yu, QIU Zheng, CHEN Chang-sheng
(College of Computer Science, Shaanxi Normal University, Xi'an 710062, China)

Abstract: CoSy is a compiler framework developed by ACE company. In order to ensure the security of inputted procedure of CoSy C compiler and avoiding the crisis of compiler's buffer overflow, model of estimating compiler's buffer overflow was put forward. According to analyzing the character of potential compiler's buffer overflow vulnerabilities in C procedure, and the model of estimating compiler's buffer overflow was founded; A method of rebuilding common CoSy medium-level intermediate representation was put forward; At last, an algorithm of estimating the security of a CCMIR procedure was put forward. It is indicated that the research can satisfactorily validate the security of the inputted procedure. And this rebuilded CCMIR model can effectively avoid the CoSy C language compiler's buffer overflow.

Key words: intermediate representation; CoSy; buffer overflow; XML

0 引 言

随着Internet的飞速发展,计算机网络安全问题日益突出。缓冲区溢出攻击成为互联网中众多系统面临的最大的挑战。

CoSy是ACE公司开发的编译器构造框架。它提供共享工具和引擎来构造编译器,编译器开发者只需专注于目标机相关程序的开发。CoSy框架生成的编译器具有可扩展性和可移植性。CoSy编译器前端产生的中间程序为通用CoSy中级中间表达CCMIR[1~3]。

为了解决缓冲区溢出问题,国内外做了大量工作,包括分离数据和控制以抵御缓冲区溢出攻击的运行环境[4~6]、AST附加安全属性分析源程序中的安全漏洞[7~9]等。以往的研究大多是在GCC编译器产生的抽象语法树上进行程序安全分析[10]。

文章分析了C源程序中缓冲区溢出漏洞的特征,给出了缓冲区溢出判断模型,同时给出了重建CoSy中间表示CCMIR的方法以及程序安全性判定算法。该方法具有可扩展性,可用于分析具有树形结构中间表示的其它编译器平台的输入程序的安全性。

1 缓冲区溢出

在造成缓冲区溢出的众多原因中,最根本原因是使用非类型安全的语言,如C语言。C语言中对字符串的操作是导致缓冲区溢出的最主要原因。C语言本质上是不安全的,它对数组和指针的操作没有自动的边界检查机制。不加检查的向缓冲区内填充大于其容量的数据,会覆盖掉其它数据。若填充的数据足够长,甚至会覆盖掉函数的返回地址。这将会导致程序产生许多不可预料的行为。对于程序中的不安全因素,应当提前发现并处理。

缓冲区问题的产生集中在存取控制上,即"已分

收稿日期:2011-11-01;**修回日期**:2012-02-05
基金项目:中央高校基本科研业务费专项资金(GK201002011)
作者简介:史 岩(1986-),男,硕士研究生,主要研究领域为Web服务;李蜀瑜,副教授,博士,主要研究领域为嵌入式系统、Web服务。

配的空间大小不能满足所要装入数据的大小”。假设已分配的空间大小为 allocated_size，要装入数据的大小为 real_size，则当 allocated_size < real_size 时，已分配空间不能满足要装入数据的需要，发生溢出。因此，可以认为 allocated_size => real_size 是防止出现此类问题的一个安全属性，只要满足这个属性，就不会产生此类问题[7,11]。

在所感兴趣的变量 v 上加入相应安全属性[v_alloc，v_len]，通过对这对整数的值域进行判断，可以确认缓冲区溢出问题。具体为变量增加安全属性的方法将在3.3节详述。

2 缓冲区溢出判断模型

基于 CoSy C 编译器的程序缓冲区溢出判断模型如图1所示。

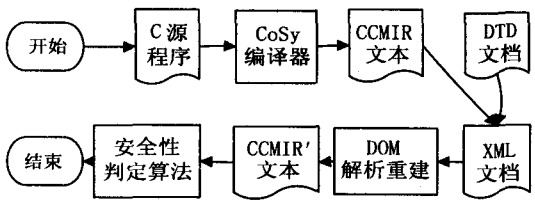

图1 缓冲区溢出判断流程

CCMIR 作为 CoSy 编译器平台输入源程序的中间表示，能够包含整个编译单元的完整表示，比较直观地表示出源程序的语法结构，对 CCMIR 遍历和操作都十分方便，而且 CCMIR 的冗余信息显然比 GCC 的抽象语法树少很多。

1) C 源程序通过 CoSy 编译器生成 CCMIR 文件1。

2) CoSy 格式的 CCMIR 通过文档类型定义(Document Type Definition，简称 DTD)定义合法的 XML 文档构建模块，产生 XML 文档。

3) 利用文档对象模型(XML Document Object Model，简称 DOM)解析 XML 文档，为 CCMIR 的结点增加安全属性，并重建 CCMIR，得到 CCMIR′。

4) 通过基于 CCMIR′的程序安全判定算法，判断程序的安全性。

3 重建 CCMIR

3.1 产生 CCMIR

通过 CoSy C 编译器 acc 对源文件 cosy.c 使用命令“./acc-dcg cosy.c”进行编译，生成的 CCMIR 文件 dumpcg1.out 如图2所示。

C 源码中的条件判断语句和循环语句，在 CCMIR 中的表现形式均为：if expression then statement1 else statement2。于是，在整个 CCMIR 代码中影响 CCMIR 结构分支的只有条件判断语句 if 和无条件转移语句 goto 语句。

CCMIR 是图形结构，不过一般可以看成特殊的二叉树。结点中存在 if 表达式，则左子树的根结点为 statement1，右子树的根结点为 statement2；结点中存在 goto 语句，则左子树的根结点为 goto 语句后紧跟的段名，右子树为空。

```
#include<stdio.h>
void func(int i);
int static stat=0;
int main()
{
  int k=8,a[2];
  while(k>0)
  {
        a[stat]=k;
    func(k);
    k--;
  }
}
void func(int i)
{
  stat=stat+1;
}
```

```
.....
arr11=array int4[0..1] sz 64 al 32
.....
STATIC DATA
  int4 : stat UE=10.00 RC=0-0
.....
END
EXPORT PROC main(OUXNRWrwA-)Impure:fun9
DECLARE
  int4: k UE=10.00 RC=100
  arr11:a UE=10.00 RC=100
BEGIN
bb0:
*cosy.c L:3 EL:5'
CST Tmp UE=1.00 RC=0
  begin
  k:=8
  goto bb1
bb1:
*cosy.c L:6 EL:6'
CST Tmp UE=1.00 RC=0
  if k^>0 then bb2 else bb4
bb2:
*cosy.c L:7 EL:9'
CST Tmp UE=1.00 RC=0
  a[stat^(*4)]:=k^
  call func(k^)
  k:=k^-1
  goto bb3
.....
```

(a) C 源程序 (b) CCMIR

图2 CoSy 编译器的输入和输出文件

3.2 CCMIR 文件转换成 XML 文档

创建 DTD 文档，在 DTD 中定义两种基本元素，node 元素和 edge 元素，分别对应 CCMIR 中的段，和段与段之间的联系。将 CoSy 的 CCMIR 文件中记录内容分为四类，制定的不同的转换规则如图3所示。

第一类：CCMIR 的标志位。每遇到一个标志位：bbx，建立一个 node 元素与之对应，node 的 id 属性值为 x；

第二类：变量声明。为每个变量建立 node 元素的子元素 variable。variable 有两个属性，name 记录变量名称，type 记录变量类型；

第三类：记录中的表达式。为每个表达式建立 node 元素的子元素 expression。expression 有两个属性，type 记录表达式类型，用来区分一般表达式和函数调用，content 记录具体的表达式内容；

第四类：记录中的跳转语句。跳转语句包含 goto 语句和 if 语句。为跳转语句建立 edge 元素，在 edge 元素中建立四个子元素，from 元素记录起始结点，to_l 元素记录左子树根结点，to_r 元素记录右子树根结点，if_

condition 元素记录跳转条件,如果不存在 if 语句,则 if _condition 的值为 1。

```
<!--结点 (node) -->
<!ELEMENT  node(variable* , expression*)>
<!ATTLIST  node  id  ID  #REQUIRED>
<!ELEMENT  variable >
<!ATTLIST  variable
name     CDATA     " "
type     CDATA     " "  >
<!ELEMENT  expression >
<!ATTLIST  expression
type     CDATA   " "
content  CDATA   " " >

<!--边 (edge) -->
<!ELEMENT  edge (from, to_l, to_r?, if_condition)>
<!ELEMENT  from  (#PCDATA)>
<!ELEMENT  to_l  (#PCDATA)>
<!ELEMENT  to_r  (#PCDATA)>
<!ELEMENT  if_condition  (#PCDATA)>
```

图 3　DTD 定义的转换规则

按照上述方法,得到图 2(b)中 CCMIR 文件的 XML 文档如图 4 所示。

```
<!--node0 bb0 -->
<node id=0>
<variable name="stat" type="int4">
<variable name="k" type="int4">
<variable name="a" type="arr11">
<expression type="common" content="k:=0">
</node>
<edge>
<from>bb0</from>
<to_l>bb1</to_l>
<if_condition> 1 </if_condition>
</edge>
<!--node1 bb1 -->
<node id=1>
<variable name="stat" type="int4">
<variable name="k" type="int4">
<variable name="a" type="arr11">
</node>
<edge>
<from>bb1</from>
<to_l>bb2</to_l>
<to_r>bb4</to_r>
<if_condition> k^>0 </if_condition>
</edge>
<!--node2 bb2 -->
<node id=2>
<variable name="stat" type="int4">
<variable name="k" type="int4">
<variable name="a" type="arr11">
<expression type="common"
          content="a[stat^{*4}]:=k^">
<expression type="call" content="func(k^)">
<expression type="common" content="k:=k^-1">
</node>
<edge>
<from>bb2</from>
<to_l>bb3</to_l>
<if_condition> 1</if_condition>
</edge>
.....
```

图 4　CCMIR 对应的 XML 文档

3.3　产生 CCMIR′

重建 CCMIR 时,为一般的变量附加属性:value,为所关注的字符串、数组、指针变量(设为 v)附加属性:v(is_global, alloc, len, is_null, is_v1_over_v2,v1_over_v2)。这样就构造起了与源程序等价而又附加了所关心安全属性的新的 CoSy 中间表达:CCMIR′。

其中,is_global 表示变量 v 是否为全局变量。其余属性的含义,以及每条表达式的执行对这些属性的影响,如引文"C/C++源程序缓冲区溢出漏洞的静态检测[7]"所述。

4　安全性判定

基于 CCMIR′的程序安全性判定算法如图 5 所示。

```
//访问结点
Visit(ElementVector){
 //根据父结点的变量ElementVector来更新当前访问结点的变量向量
 if(!UpdateCurNode(ElementVector))
 {//如果出现不安全因素，则给出错误提示
   ErrorMessage();
 }
 If(结点存在函数调用) {
  //根据被调用函数的实参初始化被调用函数树形结构的根结点
  UpdateBootNode(Parameter);
  //先序遍历被调用函数特殊二叉树结构
  PreOrderTraverse(T,ElementVector);
  //根据函数返回值和全局变量的改变，来更新当前访问结点的变量向量
  UpdateCurNode(f_return);
 }
}
//先序遍历特殊二叉树
PreOrderTraverse(T,ElementVector){
If(T){
Visit(T->ElementVector);
//结点中的条件成立，则递归遍历左子树，否则递归遍历右子树
If(expression)
 PreOrderTraverse(T->lchild, parentVector);
Else
 PreOrderTraverse(T->rchild, parentVector);
}
}
```

图 5　程序安全性判定算法

定义[12](二叉树)是一种树形结构,它的特点是每个结点至多只有两棵子树(即二叉树中不存在度大于 2 的结点),并且,二叉树的子树有左右之分,其次序不能任意颠倒。

CCMIR 树形结构的特殊性质:

1)结点带条件,先序遍历时若条件成立,则递归遍历左子树,否则,递归遍历右子树。

2)CCMIR 的二叉树带环。

5　实验结果分析

以图 2(a)中的 C 源程序为例进行分析,重建 CCMIR 得到的 main 函数的 CCMIR′树形结构如图 6 所示。

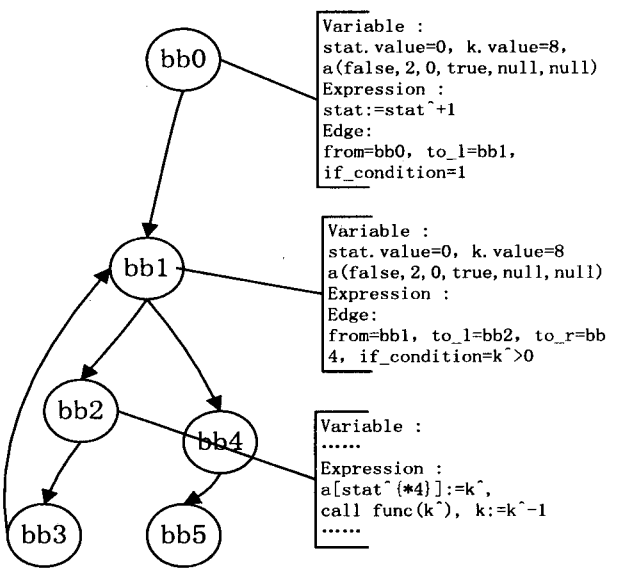

图 6　CCMIR′树形结构

使用图5程序安全性判定算法遍历树形结构：

(1)访问 bb0。stat. value=0,k. value=8,变量 a 的属性:a. alloc=2,a. len=0。转到(2)。

(2)访问 bb1。若分支条件 k^>0 满足转到(3),否则转到(5)。

(3)遍历 bb1 的左子树 bb2。一般表达式 a[stat^|*4|]:=k^,使得 a. len++;函数调用表达式 call func(k^),访问子函数 func,使得全局变量 stat. value 加1;一般表达式 k:=k^-1,使得 k. value 减1。转到(4)。

(4)访问 bb3。转到(2)。

(5)访问 bb4。转到(6)。

(6)访问 bb5,结束。

遍历结果为:在第三次执行步骤(3)后,a. len=3,即 a. len >a. alloc。属于不安全操作,程序中断,给出错误提示。

6 结束语

针对 CoSy C 语言编译器的输入程序安全性问题,提出了重建 CoSy 中间表示 CCMIR 的方法,为所关心变量附加安全属性,通过程序安全性判定算法来分析程序安全性。

该方法具有可扩展性,可用于分析具有树形结构中间表示的其它编译器平台的输入程序的安全性。

参考文献:

[1] ACE Associated Compiler Experts bv. CCMIR Primer[EB/DK]. Amsterdam,The Netherlands:[s. n.],2008.

[2] ACE Associated Compiler Experts bv. CCMIR Definition[EB/DK]. Amsterdam,The Netherlands:[s. n.],2008.

[3] ACE Associated Compiler Experts bv. CoSy Tutorial[EB/DK]. Amsterdam,The Netherlands:[s. n.],2008.

[4] 高 攀. C 语言安全编译器研究[D]. 成都:电子科技大学,2004.

[5] 魏 强,金 然,王清贤. 基于中间汇编的缓冲区溢出检测模型[J]. 计算机工程,2009(3):169-172.

[6] 匡春光,王春雷,刘 强,等. C 库中易受缓冲区溢出攻击的脆弱函数分析[J]. 微电子学与计算机,2011(2):189-192.

[7] 杨小龙,刘 坚. C/C++源程序缓冲区溢出漏洞的静态检测[J]. 计算机工程与应用,2004(20):108-110.

[8] 黄玉文,刘春英,李肖坚. 基于可执行文件的缓冲区溢出检测模型[J]. 计算机工程,2010(2):130-131.

[9] 赵奇永,郑燕飞,郑 东. 基于可执行代码的缓冲区溢出检测模型[J]. 计算机工程, 2008(12):120-122.

[10] 丁永尚,何福男. 关于缓冲区溢出漏洞的解决方法[J]. 计算机系统应用,2010(2):192-194.

[11] 王业君,倪惜珍,文伟平,等. 缓冲区溢出攻击原理与防范的研究[J]. 计算机应用研究, 2005(10):101-104.

[12] 严蔚敏,吴伟民. 数据结构[M]. 北京:清华大学出版社,2006:118-170.

(上接第92页)

(2):72-76.

[4] Löh A. Exploring Generic Haskell[D]. Netherlands:Utrecht University,2004.

[5] Pierce B C. Types and Programming Languages[R]. [s. l.]:The MIT Press,2002.

[6] Löh A, Clarke D, Jeuring J. Dependency-style Generic Haskell[C]//Proceedings of the eighth ACM SIGPLAN international conference on functional programming. [s. l.]:ACM Press,2003:141-152.

[7] Jay B. The pattern calculus[J]. ACM Transactions on Programming Languages and Systems,2004,26(6):911-937.

[8] 孙 斌. 面向对象、泛型程序设计与类型约束检查[J]. 计算机学报,2004,27(11):1492-1504.

[9] Wadler P,Blott S. How to make ad-hoc polymorphism less ad-hoc[C]//Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. [s. l.]:ACM Press,1989:60-76.

[10] Pitts A. Parametric polymorphism and operational equivalence [J]. Mathematical Structures in Computer Science, 2000 (10):231-259.

[11] Altenkirch T,McBride C. Generic programming within dependently typed programming[C]//IFIP TC2/WG2. 1 Working Conference on Generic Programming. Germany:[s. n.], 2002:1-20.

[12] Hinze R,Jeuring J,Löh A. Type-indexed Data Types:Mathematics of Program Construction[C]//Sixth International Conference,Volume 2386 of Lecture Notes in Computer Science. [s. l.]:[s. n.],2002:148-174.

[13] Lammel R,Visser J,Kort J. Dealing with Large Bananas[C]//Proc. of WGP'2000. Utrecht:Universiteit Utrecht,2000:46-59.

[14] Ehrig H. Applied and computational category theory[J]. European Association for Theoretical Computer Science,2006(6):134-135.

[15] Fegaras L,Sheard T. Revisiting catamorphisms over datatypes with embedded functions[C]//Conference Record of POPL'96:the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. [s. l.]:[s. n.],1996:284-294.

[16] 韩玉坤,王冬星. 浅谈 C++中泛型编程方法的运用[J]. 电脑学习,2007(2):47-48.

[17] 许文胜,薛锦云. 泛型编程扩展及其 JAVA 实现[J]. 计算机工程与科学,2007,29(10):89-94.

[18] 缪伟宇,邵志清. 使递归算法泛型化[J]. 计算机技术与发展,2008,18(7):96-99.