缓冲区溢出漏洞精确检测方法研究

王 雷,李 吉,李博洋
(北京航空航天大学计算机学院,北京 100083)

摘 要: 缓冲区溢出漏洞是影响系统安全性的严重问题之一,本文提出了一种利用模型检测技术对代码中潜在的缓冲区溢出漏洞进行精确检测的方法. 该方法通过静态分析,先将对缓冲区漏洞的检测转化为对程序某个位置可达性的判定. 然后,利用模型检测技术对可达性进行验证. 基于 GCC 和 Blast,我们使用这一方法构造了一个精确检测缓冲区溢出漏洞的原型系统. 最后,使用该原型系统对 wu-ftpd,minicom 和 CoreHTTP 等三个实际应用程序进行了检测,结果不仅检测出了已知的漏洞,而且发现了一些新漏洞.

关键词: 缓冲区溢出; 安全漏洞; 模型检测; 静态分析
中图分类号: TP314 **文献标识码**: A **文章编号**: 0372-2112 (2008) 11-2200-05

Precisely Detecting Buffer Overflow Vulnerabilities

WANG Lei, LI Ji, LI Bo-yang
(Computer School, Beihang University, Beijing 100083, China)

Abstract: Buffer overflow (BO) vulnerability is one of the most crucial threats to the security of software system, and a method using model checking was proposed to precisely detect potential BO vulnerabilities in source code. This method converts detecting BO vulnerabilities to verifying the reachability of certain position in programs by static analysis. Then model checking was used to do the verification job. Based on GCC and Blast, a prototype system to precisely detect BO vulnerabilities was developed for this method. At last, wu-ftpd, minicom and CoreHTTP was checked by the prototype system, which not only detected those known BO vulnerabilities but also some unknown BO vulnerabilities.

Key words: buffer overflow; security vulnerabilities; model checking; static analysis

1 引言

缓冲区溢出漏洞是最常见的软件安全漏洞之一,同时也被认为是目前最具破坏性的安全漏洞. 目前大多数快速和广为传播的蠕虫攻击,如 Code Red, SQL Slammer, Blaster 等,都利用缓冲区溢出漏洞在网络上扩散,并且造成了严重的经济损失.

目前缓冲区溢出漏洞的检测工具主要采用动态检测和静态检测两种方法. 动态检测方法在源代码中可能出现缓冲区溢出的位置插装检测代码,然后在运行中检查是否存在缓冲区溢出漏洞,例如 CCured[1] 等.

静态检测方法通过分析源代码检测可能存在的缓冲区溢出漏洞,例如字符串匹配工具 PScan[2]、ITS4[3]; 检查规格说明与实现是否相符的工具 Splint[4] 以及扩展类型检查工具 Cqual[5] 等. 但是 MIT 的论文[6] 显示,它们检测结果的误报率很高,不能实现对缓冲区溢出漏洞的精确检测.

模型检测技术[7] 是一种通过穷尽搜索状态空间,检验系统是否满足给定性质的形式化方法. 模型检测技术的优点是自动化程度高,并且在硬件系统和通信协议验证等方面的应用中取得了巨大成功. 本文拟采用模型检测技术实现对缓冲区溢出漏洞的精确检测. 但是通过对 Blast、MOPS 等模型检测工具研究与使用,发现单纯使用这类工具还不能对缓冲区溢出漏洞进行检测.

针对上述问题,我们首先对代码中缓冲区(数组、指针变量等)进行静态分析;根据分析结果,将缓冲区长度信息的变化体现在代码中,并自动在可能出现漏洞的地方添加 ERROR 标签. 这样将缓冲区溢出漏洞的查找问题转化为代码中特定位置(ERROR 标签)的可达性判定问题. 之后利用模型检测工具对特定位置的可达性进行判定. 如果判定结果为可达,则表示该位置存在缓冲区溢出漏洞. 通过这种方法使模型检测工具能够对这类漏洞进行精确检查.

收稿日期:2007-11-02;修回日期:2008-07-18
基金项目:国家 973 重点基础研究发展规划(No. 2007CB310803);教育部回国人员科研启动基金(No. 200806019);西门子中国研究院资助项目

2　精确检测方法和模型描述

我们对缓冲区溢出漏洞进行检测之前,首先需要对缓冲区溢出漏洞进行形式化的描述,然后判断代码中是否存在所描述的缓冲区溢出漏洞. 如何形式化地描述漏洞,取决于两个方面的因素:第一是该漏洞涉及到程序中哪些方面的属性;第二是使用什么方法判断源代码与这些属性的一致性. 针对第一个问题,我们分析缓冲区溢出漏洞在程序中的表现,建立相应的程序模型,根据程序模型建立漏洞模型;针对第二个问题,我们将漏洞模型转化为程序中某个位置的可达性问题,利用模型检测工具进行判断,即某个位置可达则表示该位置存在缓冲区溢出漏洞. 整个检测过程通过下面三个步骤完成:

(1) 建立程序模型. 从程序中抽象出与缓冲区溢出漏洞相关变量的属性,并抽象出这些属性的变化过程.

(2) 建立缓冲区溢出漏洞模型. 描述所关心变量的属性应该满足的约束条件.

(3) 将缓冲区溢出漏洞模型的检测转化为可达性判定问题,使用模型检测工具进行检测.

2.1　建立程序模型

缓冲区溢出漏洞可能出现在对数组元素赋值、对递引用赋值和对 strcpy 等危险函数的调用上[8]. 但是,不是所有出现这三类表达式的地方都是缓冲区溢出漏洞,还需要相应地判断数组的长度和下标的值,或者判断指针所指的缓冲区的长度,或者判断内存拷贝函数中源缓冲区的长度和目的缓冲区的长度. 这些长度信息是随着程序的控制流而变化,因此我们建立一个程序模型,该模型记录了程序中所有指针和数组所代表内存的边界信息,我们将其称为指针和数组的属性. 同时,模型还记录了属性随控制流进行变化的过程.

2.1.1　对程序中指针和数组属性的抽象

定义 1　使用 I 表示所有对内存引用的符号,包括:指针或者数组变量 I_v,自定义结构中类型为指针或者数组的域 I_f,指针类型的函数形参 I_p,指针类型的函数返回值 I_r.

定义 2　假设 $i \in I$,二元组 (min_{attr}, max_{attr}) 记录了 i 的属性,该二元组表示符号 i 所指内存的长度信息. min_{attr} 表示符号 i 所指的内存的可能最小长度,max_{attr} 表示这一内存的可能最大长度,以字节为单位. min 和 max 的下标 $attr$ 表示记录长度信息的变量名称.

定义 3　假设 $size(i)$ 表示 i 属性,$size(i) = \{(min_{attr}, max_{attr}) \mid i \in I\}$

2.1.2　对指针或数组属性更新过程的抽象

当 $\{i \mid i \in I\}$ 集合发生改变时,$size(i)$ 也会发生相应的改变. 根据使 i 发生改变的表达式,我们对 min_{attr} 和 max_{attr} 的值做相应的改变.

定义 4　把对 i 进行的操作抽象为对 $\{(min_{attr}, max_{attr}) \mid i \in I\}$ 的更新. 使用"表达式→相应更新操作"的形式表示针对不同表达式应该采取的对属性的更新操作.

根据上面的四个定义,我们使用下面四个规则对属性更新操作进行抽象.

规则 1　$\forall i \in I_v : i{+}{+} \rightarrow (min_i\text{--} \wedge max_i\text{--})$

对于对地址直接进行算术运算的表达式,对地址变量的属性施以相反的算术运算. (以为自加为例,对相应属性施以自减操作).

规则 2　$\forall i \in I_v\ \forall j \in I_v : i = j + expr \rightarrow (min_i = min_j - expr \wedge max_i = max_j - expr)$

对于对地址进行赋值操作的表达式,当右值是一个对指针变量的算术运算时,对右值属性进行相反的算术运算后赋值给左值的属性. (以加法为例,对相应属性施以减法操作).

规则 3　$\forall i \in I_v\ \forall func\_ret \in I_r : i = func\_ret \rightarrow (min_i = min_{func\_ret} \wedge max_i = max_{func\_ret})$

对于调用自定义函数的表达式,把函数返回值当作普通变量对待. 将返回值的属性传递给等号左值变量的属性.

规则 4　$\forall i \in I_v\ libfunc\_ret \in I_r : i = libfunc\_ret \rightarrow (min_i$ 和 max_i 根据实际函数作特殊处理).

对于分配内存的库函数的调用表达式,例如 malloc,min 设置为 0,max 设置为分配内存的大小. 对于获取外界输入的库函数的调用表达式,例如 getenv,min 设置为 0,max 设置为 $+\infty$.

2.2　建立缓冲区溢出漏洞模型

在程序模型中添加了与缓冲区溢出漏洞相关变量的属性之后,我们在程序中对数组元素赋值、对递引用赋值和对 strcpy 等危险函数调用之前对所关心变量的属性进行限制. 我们将这一限制描述为一条包含指针或数组属性的不等式. 如果代码不能满足这一限制,则可以确定为存在漏洞. 一个程序中所有限制的总和称为这个程序的缓冲区溢出漏洞模型.

针对危险函数,可以构造出如下限制条件:

(1) $func(obj, src, n); \rightarrow (min_{obj} > n)$

对于确定长度的内存拷贝函数,例如 memcpy(…)、strncpy(…),对其参数的限制可以简单表示为拷贝到目的内存的内容长度不能超过该内存可能的最小长度.

(2) $func(obj, src); \rightarrow (min_{obj} > max_{src})$

对于字符串拷贝函数,例如 strcpy(…),拷贝到目的内存的内容的长度由字符串结束符的位置决定. 我们无法了解具体字符串的长度,因此采取积极的策略,假定源字符串的长度为可能的最大长度. 另外,在调用 strcpy

之前,我们可以认定源字符串的内容不会超过其最大长度,如果强制为字符数组赋值使字符串超过数组的长度的话,则在赋值操作时便可查出该漏洞.

2.3 漏洞模型转化为可达性判定问题

对于漏洞模型中的每一条限制,我们在该限制对应的位置前插入一条 if 语句,将漏洞模型中对相关变量属性的限制作为 if 语句中的条件. 例如,对于语句 $*p = 'a'$,漏洞模型中对 p 的属性限制为 $(min_p > 0)$,则将此限制转换为伪代码如图 1 所示.

```
1  if (min_p <= 0) {
2          ERROR:  goto ERROR;
3  }
```

图 1 可达性判定问题示例

通过这样的变化,我们将对漏洞的检测转化为了对代码中 ERROR 标签的可达性的判定问题. 如果 ERROR 标签位置可达,则说明对属性的限制在程序中被违背,存在一个漏洞.

3 原型系统描述

根据上面的方法,我们建立了一个对源代码中缓冲区溢出漏洞进行精确检测的原型系统. 该原型系统由前端分析工具,后端模型检测工具以及连接前端后端的代码处理工具组成. 前端对源代码中与指针、数组以及危险函数相关的代码部分进行分析. 代码处理工具根据分析结果,采用 source-to-source 模式将附加代码插入到源代码中. 后端使用模型检测工具 Blast 对处理后的代码进行检测. 该原型系统的输入包括源代码和错误描述(目前只能检测缓冲区溢出漏洞),输出是缓冲区溢出漏洞在源代码中的位置,以及产生漏洞的执行路径. 系统结构如图 2 所示.

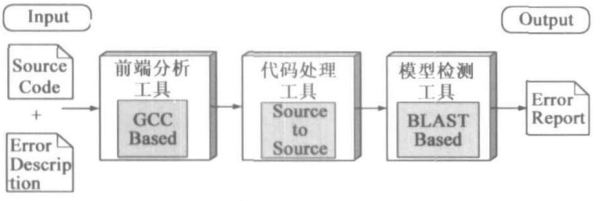

图2 原型系统框架

3.1 基于 GCC 的前端分析工具

前端分析工具在 GCC-4.0.0 基础上开发,对 GCC 建立的抽象语法树(Abstract Syntax Tree,AST)进行分析. 前端对指针和数组变量的声明、赋值、访问,以及危险函数的调用进行分析,搜集代码中与指针数组以及危险函数相关的信息,并判断应该对代码中的变量进行什么样的限制.

3.2 代码处理工具

代码处理工具根据前端对源码的分析,将源代码中变量属性和对属性的限制转化成 C 代码,并插装到源码中. 插装的内容包括三部分:(a). 记录指针和数组属性的变量;(b). 属性变量的传递;(c). 在代码特定位置上这些属性需满足的约束条件. 代码的插装规则如表 1 所示.

表 1 模式行为规则表

Pattern	Action
$p_0 = p_1;$	$set\_length(p_0, p_1)$
$p_0 = p_1 \pm c;$	$set\_length(p_0, p_1 \mp c)$
$^*p_0 = x;$	$build\_assert(p_0, 0)$
$^*(p_0 \pm c) = x;$	$build\_assert(p_0 \mp c, 0)$
$a[i] = x;$	$build\_assert(i, a)$
$a[i \pm c] = x;$	$build\_assert(i \pm c, a)$
$func1$(dst,src);	$build\_assert$(dst,src)
$func2$(dst,src,n);	$build\_assert$(dst,n)
$func3$(ptr,…);	$set\_param\_length(func3$,ptr)

表 1 中左边一列表示可能出现漏洞的模式和传播过程,右边一列表示应该采取的相应动作. 其中 $p_0 = p_1$、$p_0 = p_1 \pm c$ 分别表示将一个指针 p_1 或者一个指针运算表达式 $p_1 \pm c$ 的结果赋给指针 p_0. 相应的动作为记录 p_0 指针长度的变化(set_length). $^*p_0 = x$ 和 $^*(p_0 \pm c) = x$ 是对 p_0 或 $p_0 \pm c$ 所指地址的赋值操作,因此通过 $build\_assert$,在它们所指向的空间长度小于等于 0 时,插入 ERROR 标签. 对数组赋值 $a[i] = x$ 或 $a[i \pm c] = x$,与前一种情况类似. 最后是几种特殊的函数:$func1$ 表示类似于 strcpy 这样的危险函数,$func2$ 表示类似于 memcpy 这样的危险函数,它们分别需要根据源和目的缓冲区的长度,通过 $build\_assert$,在对目的缓冲区的操作越界时,插入 ERROR 标签;$func3$ 表示一个或多个指针作为函数的参数传递到另一个函数当中,处理这种跨函数的问题,需要为所有参数是指针类型的函数,生成额外的用于记录参数指针长度的信息(set_param_length).

采用 source-to-source 模式的好处在于插装代码后得到的结果是可以编译的 C 语言代码,可以在其上进行二次分析和优化处理. 而且,后端使用的模型检测工具可以轻松替换,只要支持对 C 源码进行检查的工具都可以作为后端使用.

3.3 模型检测工具 Blast

Blast 是一个控制流敏感、且支持过程间分析的模型检测工具,由 UC Berkeley 的 Henzinger 等人开发. Blast 可以进行代码的可达性分析,也就是判断程序是否可以从入口处开始执行,并到达某个指定的位置.

因此,我们使用 Blast 检测程序是否可以到达插装

的 ERROR 标签处.如果可达,则表示发现了一个漏洞.对于漏洞产生的原因,用一条从程序入口到漏洞产生点的执行路径来表示.

4　实验结果

以上原型系统已经在 Red Hat 9(2.4.20)和 GCC4.0 平台上实现.为考查本文检测方法对实际应用程序的检测能力,我们选择了 wu-ftpd-2.6.2,minicom-1.80 和 CoreHTTP 0.5.3 等三个实际的程序进行了测试.测试结果如表 2 所示.文[9]发现 wu-ftp2.5.0 中有 1 个缓冲区溢出漏洞,本文工具针对 wu-ftpd-2.6.2 检测出了 3 个实际存在的漏洞.文[10]发现了 minicom 中 1 个漏洞,本文工具发现了 3 个漏洞.2007 年 7 月份绿盟网站[11]发现了 coreHTTP 中 1 个漏洞,本文工具检测出了 8 个漏洞.

表 2　实际程序的测试数据

程序名	代码行数	发现的漏洞数
Wu-ftp	16528	3
Minicom	6236	3
CoreHTTP	5008	8

5　相关研究工作

20 世纪 80 年代 E. M. Clarke、E. A. Emerson、J. P. Quielle J. Sifakis 等人提出了模型检测技术,该方法自提出以来在硬件系统和通信协议验证等方面的应用中取得了成功.目前模型检测技术在软件检测方面的应用已经成为了研究热点.UC Berkeley 的研究小组开发了 MOPS[12]和 Blast[13].MOPS 是 Chen 和 Wagner 在 2002 年的合作项目,部分解决了模型检测不能应用于大型系统的问题,但是代价是损失了检测精度.Henzinger 小组开发的 Blast 针对模型检测中状态空间爆炸问题,采用 Lazy Abstraction 算法指导模型检测中对反例的搜索.Stuttgart University 的 Kiefer 研制了工具 Moped[14],并在数学上进行了一系列证明下推自动机正确性的研究.CMU 的 Clarke 领导开发了 SatMate,VCEGAR ,CBMC,MAGIC,smv2vcd,SyMP,BMC,SMV 等一系列各具特色的模型检测工具*.除了各知名大学以外,贝尔实验室也推出了两个模型检测工具 SPIN[15]和 Verisoft[16].微软研究院的 Thomas Ball 领导开发了 SLAM 系列工具[17].

虽然模型检测在软件检测方面的研究非常活跃,但是专门针对缓冲区溢出漏洞的模型检测方法研究并不多见.CMU 的 Sagar Chaki 和 Scott Hissam[18]提出了利用模型检测精确查找缓冲区溢出漏洞的想法,但并没有给出任何实现方案.Wagner[19]使用整数范围(integer range)来描述指针或缓冲区,通过对整数范围的分析,检查缓冲区溢出漏洞.但是该方法实际上是流不敏感的静态分析方法,因此误报率较高.

6　结论

本文将编译系统中的静态分析技术与模型检测技术相结合,将缓冲区溢出漏洞检测问题转换为可达性判断问题,解决了模型检测工具不能直接应用于缓冲区溢出漏洞检测的问题.与传统静态分析工具相比,本文实现的原型系统具有检测精度高,低误报率等优点.但是,对于大规模或带有复杂控制流的代码,检测过程中存在状态空间爆炸的问题,一些 ERROR 标签不能在有限时间内完成检测.另外,原型系统不支持对函数递归调用的检测.

将来我们准备进一步挖掘编译技术中可以利用的地方,比如常量折叠等优化技术,充分利用这些技术有效减少待检测系统的复杂度.另外,也可以考虑使用程序切片技术降低待检测系统的规模,减少系统模型的状态空间.

参考文献:

[1] G C Necula, S McPeak, W Weimer. CCured: typesafe retrofitting of legacy code [A]. ACM SIGPLAN-SIGACT Conference on the Principles of Programming Languages (POPL) [C]. Portland: ACM Press, 2002. 128 - 139.

[2] U Hermann. Overview of pscan source package [EB/OL] http://packages.qa.debian.org/p/pscan.html. 2006 - 11.

[3] J Viega, J T Bloch, T Kohno, G. McGraw. ITS4: a static vulnerability scanner for C and C++ code [A]. 16th Annual Computer Security Applications Conference [C]. New Orleans, US: IEEE, 2000. 245 - 257.

[4] D Evans, D Larochelle. Improving security using extensible lightweight static analysis [J]. IEEE Software, 2002, 19(1): 42 - 51.

[5] U Shankar, K Talwar, J S Foster, D Wagner. Detecting format string vulnerabilities with type qualifiers [A]. Proc of the 10th USENIX Security Symposium [C]. Washington, DC: USENIX Association, 2001. 16 - 16.

[6] M Zitser, An Evaluation of Static Source Code Analyzers [D]. Massachusetts Institute of Technology, 2003.

[7] E Clarke, O Grumberg, D Peled, Model Checking [M]. Massachusetts: The MIT Press, Cambridge, 1999.

[8] O Lichtenstein, A Pnueli. Checking that finite state concurrent programs satisfy their linear specification [A]. Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages [C]. New Orleans, US: ACM, 1985. 97 - 107.

[9] D Larochelle, D Evans. Statically detecting likely buffer over-

* http://www.cs.cmu.edu/～modelcheck/

flow vulnerabilities [A]. 2001 USENIX Security Symposium [C]. Washington, DG, 2001. 13 - 17, 177 - 190.

[10] Matt Conover, w00w00 on Heap Overflows [EB/OL]. http://www.w00w00.org/files/articles/heaptut.txt, 1999.

[11] NSFOCUS, CoreHTTP http.c buffer overrun [EB/OL]. http://www.nsfocus.net/vulndb/10719, 2007.

[12] H Chen, D Wagner. MOPS: An infrastructure for examining security properties of software [A]. Proc of the 9th ACM Conf. on Computer and Communications Security (CCS) [C]. Berkeley, US: University of California at Berkeley, 2002. 235 - 244.

[13] T Henzinger, R Jhala, R Majumdar, G. Sutre. Lazy abstraction [A]. Proc of the 29th Annual Symp. on Principles of Programming Languages (POPL) [C]. Portland, US: ACM, 2002. 58 - 70.

[14] S Kiefer. Abstraction Refinement for Pushdown Systems [D]. Universit ? t Stuttgart, 2005.

[15] G J Holzmann. The model checker SPIN [J]. IEEE Trans on Softw Eng, 1997, 23(5): 279 - 295.

[16] P Godefroid. Model checking for programming languages using VeriSoft [A]. Proceedings of the 24th ACM Symposium on Principles of Programming Languages [C]. Paris, France: ACM, 1997. 174 - 186.

[17] T Ball, S Rajamani. The SLAM project: debugging system software via static analysis [A]. POPL 2002 [C]. Portland, Oregon: ACM, 2002. 1 - 3.

[18] Sagar Chaki, Scott Hissam. Precise Buffer Overflow Detection via Model Checking, White paper [EB/OL]. http://www.sei.cmu.edu/staff/chaki/publications/WhitePaper-2005-2.html, 2005.

[19] Wagner D, Foster J, Brewer E, et al. A first step towards automated detection of buffer overrun vulnerabilities [A]. Proc of Symposium on Network and Distributed Systems Security [C]. San Diego, CA, 2000, 02: 3 - 17.

作者简介:

王 雷 男, 1969 年生于黑龙江, 北京航空航天大学计算机学院副教授, 博士, 主要研究领域为编译技术, 模型检测技术和操作系统.
E-mail: wanglei@buaa.edu.cn

李 吉 男, 1979 年生于四川, 北京航空航天大学计算机学院硕士研究生, 主要研究领域为编译技术和模型检测技术.

李博洋 男, 1982 年生于北京, 北京航空航天大学计算机学院硕士研究生, 主要研究领域为编译技术和模型检测技术.