第 36 卷 第 2 期 计 算 机 工 程 2010 年 1 月
Vol.36 No.2 Computer Engineering January 2010
·安全技术· 文章编号：1000—3428(2010)02—0130—02 文献标识码：A 中图分类号：TP393.8

基于可执行文件的缓冲区溢出检测模型

黄玉文[1,2]，刘春英[1]，李肖坚[2,3]
(1. 菏泽学院计算机与信息工程系，菏泽 274015；2. 广西师范大学计算机与信息工程学院，桂林 541004；
3. 北京航空航天大学北京市网络技术重点实验室，北京 100083)

摘 要：给出缓冲区溢出的基本原理和现有检测技术，针对二进制可执行文件中存在的缓冲区溢出漏洞，提出一种缓冲区溢出检测模型，该模型采用静态检测和动态检测相结合的方法。对检测结果采取污点跟踪法进行人工分析，采用插件技术给出缓冲区溢出检测模型的具体设计。实验结果证明该模型的设计是有效的。

关键词：缓冲区溢出；可执行文件；静态检测；动态检测；人工分析

Detection Model for Buffer Overflow Based on Executable File

HUANG Yu-wen[1,2], LIU Chun-ying[1], LI Xiao-jian[2,3]
(1. Computer and Information Engineering Department, Heze University, Heze 274015; 2. College of Computer Science and Information Engineering, Guangxi Normal University, Guilin 541004; 3. Beijing Key Laboratory of Network Technology, Beihang University, Beijing 100083)

【Abstract】This paper describes the basic principles of buffer overflow and the current detection technology, and presents a detection model for buffer overflow vulnerability on binary executable file. The model uses static and dynamic detection technologies, and analyzes artificially the result in stain-tracking way. It gives specific reality for the detection model for buffer overflow in plug-in technology. Experimental results show effectiveness of the defection model.

【Key words】buffer overflow; executable file; static detection; dynamic detection; artificial analysis

1 概述

据 CERT 统计，自 1995 年到 2006 年漏洞报告累计达到 30 780 个，其中，2006 年共报告漏洞 8 064 个，平均每天超过 22 个。在过去 10 年中，缓冲区溢出是安全漏洞最常见的一种形式。与此同时，网络安全形势日趋严峻，大规模互联网攻击事件频繁发生。在这些攻击事件中，最常见的攻击是针对系统和程序自身存在的漏洞编写的攻击程序，其中又以属于堆栈溢出的缓冲区溢出漏洞为主[1]。缓冲区溢出防御的研究得到不少研究机构的重视，但目前大多是基于源代码的研究，基于二进制代码的研究尚处于起步阶段。

本文提出了一种基于二进制可执行文件的缓冲区溢出检测模型，并对模型进行了设计和实现。

2 缓冲区溢出基本原理和现有检测技术

缓冲区是程序运行时在内存中临时存放数据的地方。当程序试图存放的数据块大小超过程序事先申请到的内存缓冲区大小时，会发生缓冲区溢出[2]。程序在运行中发生缓冲区溢出现象，很大程度上是因为在很多程序中缺乏对运行库函数的缓冲区边界检查，人们把缓冲区溢出的原因归结到这类函数大量存在并广为使用上。本文讨论的二进制可执行代码的缓冲区溢出检测主要针对 C/C++语言。C/C++语言自带很多库函数，出于效率的考虑，这些库函数中的字符串操作函数、打印类函数在实现时没有加入边界检查代码。这类函数使用频度很高，很容易出现缓冲区溢出的情况，因此，这类库函数习惯上称作危险函数。

目前，缓冲区溢出检测技术主要分为静态检测和动态检测 2 类。

静态检测主要通过分析程序的源代码或二进制代码的反汇编结果检测程序中存在的缓冲区溢出漏洞[3]。优点是不需要执行程序，查找速度快，适合自动化检测。缺点是适应性不强，存在大量误报。常用的静态检测工具有 ITS4, Flawfinder, Splint 和 BOON。动态检测是在程序的运行过程中，通过监测程序的运行状态来检测是否发生了缓冲区溢出[4]。优点是准确率高、针对性强，缺点是系统开销大、技术复杂，对分析人员要求高。常用的动态检测工具有 StackGuard, ProPolice, FormatGuard。

3 缓冲区溢出检测模型

缓冲区溢出模型首先对二进制可执行文件进行静态检测，然后对静态检测的结果进行动态检测和人工分析，以提高检测的准确率。该模型主要包括 3 个模块：静态检测模块，动态检测模块和人工分析模块，见图 1。

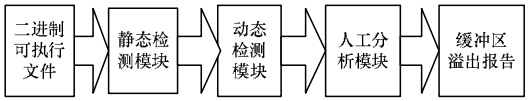

图 1 缓冲区溢出检测模型

3.1 静态检测模块

静态检测模块首先对二进制可执行文件进行反汇编，然后对汇编代码中危险函数的普通形式和展开形式进行识别，最后对识别的危险函数进行缓冲区溢出判定，生成候选缓冲区溢出报告。其模块如图 2 所示。

基金项目：国家部委基金资助项目
作者简介：黄玉文(1978－)，男，硕士研究生，主研方向：网络安全；刘春英，讲师、硕士；李肖坚，副教授、博士
收稿日期：2009-06-26 **E-mail：**ywhuang@mailbox.gxnu.edu.cn

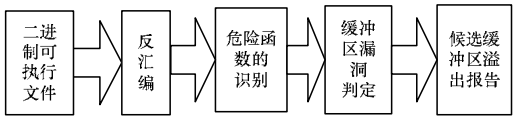

图 2　静态检测模块

(1)危险函数的普通形式识别

危险函数的普通形式在反汇编代码中以相应函数名的形式出现，所以，只需把所识别的反汇编代码中的函数名与危险函数列表中的函数名进行比较，如果存在，该处代码前的地址就是危险函数所在行地址，记下该地址、危险函数名以及此函数被何函数调用及其所在行的汇编代码。

(2)危险函数的展开形式识别

编译器出于对所编程序优化的目的，为了提高程序运行时的效率，经常在目标代码中出现函数的展开形式。通过查看这些危险函数展开后的汇编指令，发现有些指令在不同编译环境中都会出现，本文把这类指令称为必然指令，对危险函数展开形式的识别就转换为对必然指令的识别。

(3)缓冲区溢出的判定

通过危险函数普通形式和危险函数展开形式的识别，找到了可能的溢出点，因为并不是危险函数的所有调用都会发生缓冲区溢出，所以还要对查找到的危险函数进行缓冲区溢出判定。在进行缓冲区溢出判定时，本文把危险函数分为如下 3 类：

1)外界输入类函数，如 scanf(), gets()。此类函数只要在程序中使用，可能就会引起缓冲区溢出。

2)格式化串类函数，如 printf(), vprintf()。对于这类函数，要查看其参数是否含有外界输入的字符。如果参数中含有外界输入的字符，则可能引起缓冲区溢出。

3)字符串处理类函数，如 strcpy(), strcat()。对于此类函数，要查看其传入参数所代表的空间大小，当源操作空间大于目标操作空间时，此类危险函数可能引发缓冲区溢出。

根据危险函数所在位置，采取逆向分析，找到并分析传递给此危险函数的参数，判定是否发生缓冲区溢出。

3.2　动态检测模块

动态检测模块是缓冲区溢出检测模型的核心，动态检测模块利用静态检测的结果，参照函数调用关系图(FCG)和控制流图(CFG)，将测试数据注入二进制可执行文件中。通过设置断点，单步执行程序，动态监视运行时堆栈内数据的变化。在动态检测中，测试数据和监视对象的选择很重要。要多次运行程序，注入不同的测试数据，尤其是能够引起缓冲区溢出的数据。程序运行时一般在进程的堆栈内开辟数据空间，监视数据空间在危险函数运行前后的变化状况，以此判断是否发生缓冲区溢出。

3.3　人工分析模块

人工分析模块结合程序的控制流图和函数调用关系图，利用人工分析的方法对静态检测到的缓冲区溢出发生点进行确认。在软件与用户交界处最有可能引起漏洞，在逆向工程中，输入追踪的方法有很广泛的应用。人工分析模块是缓冲区溢出检测的最后一个环节，也是最重要的一环。发生缓冲区溢出的最根本原因是有外界输入的数据，如果外界输入的数据不会到达危险函数所在位置，则此位置不会真正发生缓冲区溢出。人工分析模块通过分析外部数据是否到达候选缓冲区溢出报告所记录的位置，以此判定是否发生缓冲区溢出。在本文中，外界输入的数据又称为污点数据。

4　缓冲区溢出检测模型具体设计

在模型具体实现时，本文采用逆向工程的主流开发平台——IDA Pro。IDA Pro 提供了各种 API 及 SDK，用户可以根据自己的需求编写各种插件并利用 IDC 脚本扩充其功能。IDA Pro 从 4.17 版开始提供了流程图功能，可以使用其图形功能生成控制流图和函数调用关系图，供动态检测和人工分析使用。

4.1　静态检测的设计

采用 IDA 插件实现静态检测，静态检测插件包含 5 个部分：预处理，初始化函数，清除函数，主体运行函数和辅助说明部分，其中，主运行函数主要包括以下几部分：

```
void IDA_staticdetect_run (int arg){
find_usu_func( );       //危险函数的普通形式识别
find_unflod_func( );    //危险函数的展开形式识别
judge_func_vuler( );    //缓冲区溢出漏洞判定
}
```

在进行缓冲区溢出判定时，分析危险函数的参数所代表空间大小非常重要，对于危险函数的不同分类，要用不同的方法对参数进行分析，危险函数的参数分为如下 3 类：

(1)形如[ebp+var_xx]或[esp+var_xx]的操作数，表示操作数为堆栈内所开辟的空间，此时要分析反汇编代码中的声明区域，通过计算得到缓冲区大小，以此判断参数所代表空间的大小。

(2)形如[ebp+arg_xx]的操作数，表示操作数是从函数外传递过来的参数，不是栈中所开辟的空间，此时要通过计算外部函数中的堆栈缓冲区大小判定参数所代表空间的大小。

(3)形如[offset xxxx]的操作数，代表全局变量空间或常量所占空间，此时要计算全局变量或常量所占空间的大小。

通过比较危险参数所代表空间大小，进行缓冲区溢出判定，把发生缓冲区溢出的相关信息保存在缓冲区溢出列表文件中作为候选缓冲区溢出报告，供动态检测和人工分析使用。

4.2　动态检测的设计

在具体设计时，采用 IDA 插件技术实现动态检测，把动态检测设计为一个具有记录功能的调试器，其主运行函数如下：

```
void IDA_dynamicdetect_run(int arg){
while(!feof(fp)){
set_breakpoint();
run_to_breakbefore(); //运行程序到断点前
get_breakbefore_infor();
run_breakpoint(); //单步运行断点处程序
get_breakafter_infor();
compare_infor(); //比较断点前后堆栈内信息
}}
```

选择断点处危险函数参数所代表空间的前一位置的内容作为记录对象，通过比较此内容在危险函数执行前后是否发生变化，确定断点所在处的危险函数是否发生缓冲区溢出，并记录检测结果。

4.3　人工分析

进行人工分析时选用 IDA Pro 自带的调试器进行污点数据跟踪。其步骤如下：(1)选择污点数据注入点。把外界输入类函数所在的位置作为污点数据注入点。(2)在指令追踪模式下运行程序。(3)查看污点数据流向。在指令跟踪窗口中，输入污点数据的地址，查看静态检测生成的候选缓冲区溢出报

(下转第 134 页)