第 32 卷 第 18 期 计 算 机 工 程 2006 年 9 月
Vol.32 № 18 Computer Engineering September 2006
·安全技术· 文章编号：1000—3428(2006)18—0141—03 文献标识码：A 中图分类号：TP309

# 基于二进制代码的缓冲区溢出检测研究

叶永青，李 晖，郑燕飞，洪 璇，郑 东
(上海交通大学密码与信息安全实验室，上海 200030)

**摘 要：**随着 Internet 应用的广泛深入，计算机系统的安全问题日益引起人们的重视，其中，缓冲区溢出漏洞攻击的数量呈逐年上升之势。该文从缓冲区溢出的原理开始，描述了一种利用静态分析和动态分析相结合的基于二进制代码的缓冲区溢出分析检测技术及工具，比较和分析了该工具检测二进制代码的结果与传统工具检测对应源程序的结果，并提出了存在的不足和改进之处。

**关键词：**缓冲区；缓冲区溢出；静态分析；动态分析

# Analysis of Buffer Overflow in Binary Files

YE Yongqing, LI Hui, ZHENG Yanfei, HONG Xuan, ZHENG Dong
(Lab of Cryptography & Information Security, Shanghai Jiaotong University, Shanghai 200030)

**【Abstract】** With Internet goes further, people pay more and more attention to computer security problems. And among them, the number of buffer overflow attacks is growing by year. This article begins with the theory of buffer overflow attacks, describes a method and a tool using static analysis and dynamic analysis to detect buffer overflow in binary files, compares and analyses the result of running this tool and traditional buffer overflow detect tool and indicates the defects of the tool and how to improve it.

**【Key words】** Buffer; Buffer overflow; Static analysis; Dynamic analysis

## 1 研究背景

近年来，由系统安全漏洞导致的网络攻击越来越多，根据 Symantec 的《互联网安全威胁报告》，2003 年，软件公司和安全研究员共向公众发布了 2 636 个软件安全漏洞，关于缓冲区溢出漏洞占 56.76%以上。

因此，如何发现存在于系统软件、关键应用软件、关键设备里的未公开漏洞是一项具有挑战性的工作。目前，国内外对这方面的研究还十分有限。绝大部分的研究工作都集中在如何对现有、已被发现的系统漏洞进行防治。缓冲区溢出攻击防范和检测工具大多只是针对某种攻击目标的解决方案，适用性有限。此外，绝大多数方法是基于源代码的，这些无疑都极大地限制了现有工具的使用范围。

那么，是否有方法能在某种程度上突破这些限制，在不给出源代码的情况下，对二进制文件进行分析和检测，本文就此作了一些探讨，研究并提出了一套针对应用程序的二进制代码进行程序分析并发现隐藏缓冲区漏洞的方法，用程序实现该方法，并对结果进行了比较。

## 2 原理以及国内外研究现状

### 2.1 缓冲区溢出攻击的原理

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了程序事先申请到的内存缓冲区大小时就会发生缓冲区溢出。如果溢出的数据块恰好覆盖与缓冲区相邻的子程序或函数返回地址，而覆盖这一地址的又恰好是一个程序的入口，那么这一程序将自动运行。攻击者可以借此精心构造填充数据，让程序转而执行特殊的代码，最终获得系统的控制权或取得其他非法权益。

### 2.2 防范和检测技术的研究现状

目前针对缓冲区溢出漏洞的攻击和防范技术分析主要分为 2 类：静态分析和动态分析。

(1)静态分析：主要通过对源代码的分析和审计，判断缓冲区溢出漏洞，如 Lclint，RATS，以及我们主要参考的 Wanger 的方法等

(2)动态分析：通过改变运行环境或者系统功能，使程序避免发生缓冲区溢出攻击在程序执行过程中，采用压力测试等方法，查找缓冲区溢出漏洞，如 FIST，S-Tool。

静态测试的方法能够有效地发现一些常见的缓冲区溢出漏洞，而且可以检测大规模的程序。其缺点是不够灵活，适应性不强，必须不断更新程序样本漏洞数据库，而且其误报率较高，难以准确判断缓冲区溢出的漏洞。它需要依赖于程序源代码，这样对于一些不公开源代码的商业软件就难以使用。动态测试通过自动化工具，自动生成测试数据，通过仿真攻击状态下应用程序执行来判断缓冲区溢出漏洞位置。其缺点是，它只能防范已知的攻击方法，而不是根除实际的安全漏洞。其次，它会使系统开销增大。虽然动态测试方法只需要基于二进制代码，但是其效率不高，而且需要预先估计程序漏洞位置，以及相应的输入和输出。因此，传统的缓冲区溢出漏洞检测手段难以有效地检测基于二进制代码的系统及应用软件中的安全漏洞。

目前较著名的缓冲区溢出漏洞检测工具有：StackGurad, LCLint, StackShield 等。

## 3 研究方案及结果

本文的缓冲区溢出漏洞挖掘模型主要分为 3 个模块：程序解析模块，静态漏洞分析模块和动态测试模块。

**作者简介：**叶永青(1980—)，男，硕士生，主研方向：信息安全；李 晖，博士生；郑燕飞，讲师；洪 璇，博士生；郑 东，副教授

**收稿日期：**2005-10-12 **E-mail：**yeyounqing@sjtu.edu.cn

程序解析模块以被分析程序的可执行二进制文件为输入，该模块首先用 Linux 下 objdump 和 nm 命令对可执行文件的依赖关系进行分析，生成文件依赖关系图。然后利用 IDA 这样的反汇编工具生成相应的汇编代码。最后，程序解析模块解析汇编代码，生成函数调用关系图和函数内部的控制流图(见图 1)。

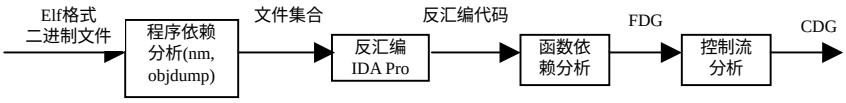


**图1 缓冲区漏洞挖掘模型的程序解析模块**

在对程序结构进行解析后，静态分析模块根据函数调用关系和控制流来顺序分析汇编代码。在静态分析中，我们将缓冲区问题影射为整数限制问题，结合图论方面的知识，抽象汇编中的特征代码，将缓冲区溢出的问题抽象为一个整型数据范围的分析。并生成缓冲区溢出漏洞分析报表，对潜在的漏洞代码给出警告(Warning)信息。

动态分析是在静态分析的基础上，对所有潜在漏洞处进行运行期代码测试(runtime code-testing)，采用 FIST 等工具对潜在漏洞处注入长字符串，并观察函数栈状况，从而进一步判断缓冲区溢出漏洞，过滤在静态分析中错误判断的缓冲区溢出漏洞，并给出完整的漏洞分析报告。

### 3.1 程序解析模块

程序解析模块主要由程序依赖分析、反汇编、函数依赖分析和程序流程分析 4 部分组成。程序解析模块对 Linux 下 ELF 格式的可执行文件进行初步的分析，解析其结构从而便于静态分析模块能够高效准确地分析文件中存在的可能的缓冲区溢出漏洞。

(1)程序依赖关系分析

对于一个程序的分析，不能仅仅分析其主程序，其相关的库函数也是分析的目标，因此对于分析的第 1 步就是确定文件的相互依赖关系。

我们利用 Linux 中的 objdump 和 nm 命令来判断文件的相互依赖关系。利用 objdump –R 指令来查看一个可执行文件中调用了哪些外部函数。也可以利用 Linux 下的 nm 命令来查找目标文件中定义实现了哪些函数，从而结合 objdump 命令中找到的函数调用关系建立文件相互的依赖关系。

(2)反汇编工具 IDA pro

IDA Pro 是 DataRescue 开发的专业反汇编工具，它支持多处理器(Multi-Processor)、多文件格式， 在 Windows 上也可以反汇编 Linux 的 ELF 格式文件。我们用 IDA Pro 对需要分析的程序二进制代码进行反汇编，生成所需的汇编代码，用于进一步的漏洞分析。

(3)函数依赖分析和控制流分析

本模块分为 2 个子部分，即函数依赖分析和控制流分析。函数依赖分析主要是得到子过程调用的关系图，以便后续的分析。每一个子过程作为图中的一个节点，根据调用关系以 main 函数为根得到一个函数依赖关系图(FDG)。

对于函数依赖图中，除了标准 C 库函数节点以外，还要对每一个函数节点作进一步的控制流分析。基于控制流的分析能够有效地提高漏洞发掘的准确性。对于汇编代码中的控制流的分析，我们是根据各种 jmp 指令来做一个近似的判断，从而生成一个函数内控制流图(CDG)，帮助提高下一步的静态分析的准确性。

### 3.2 静态分析模块

静态分析模块根据上面函数依赖和控制流分析模块生成的 FDG 和 CDG 来顺序分析整个程序。静态分析模块将缓冲区溢出问题首先转换为整型数据范围的分析，然后结合图论的方法进行缓冲区漏洞挖掘。其基本思想如下：

对所有的缓冲区建模，看作由 2 个整型数据构成的一个“范围”。不必跟踪字符串变量的真实内容，而是给每个缓冲区赋两个权值：给它分配了多少字节大小，以及它实际使用了多少字节大小，也就是字符串实际的长度。这样，问题就转换成了只要检查每个字符串，它实际使用的字节数是不是超过了给它分配的大小。

静态分析模块也分为 3 个子模块：缓冲区边界判定，整数限制生成模块和限制分析模块。缓冲区边界判断主要是区分汇编中的变量是否为一缓冲区，并判断其大小。整数限制生成通过约束生成语言将汇编中某些操作缓冲区的代码影射为整数限制。在生成整数限制后，分析模块利用图论的知识分析这些整数限制，生成警告信息(见图 2)。

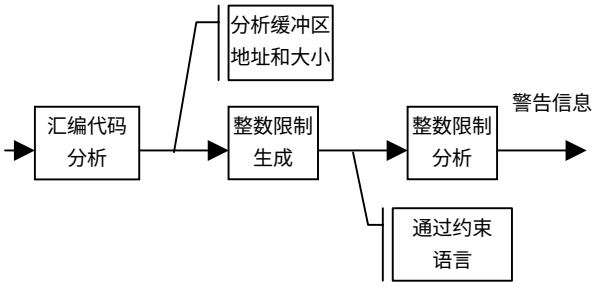


**图 2 缓冲区漏洞挖掘模型的静态分析模块**

(1)缓冲区边界判断模块

针对汇编代码的缓冲区大小的判断不像基于源代码的分析那么方便，不过也可以采用一些方法判断缓冲区的大小，我们判断缓冲区大小的基本思想是根据指令对内存单元访问情况来界定缓冲区边界。可以通过分析未直接访问的堆栈单元、局部变量的访问方式或其他方法来确定目标缓冲区长度。

(2)整数限制生成

基于 C 和 C++语言的代码中，大多数对于缓冲区的操作都是通过一些标准 C 库函数实现的。这些库函数可以为对缓冲区进行分配空间、赋值、改变大小等操作。因此这些标准 C 库函数可以映射成对于缓冲区整数的限制。我们定义了从 C 到整数限制的约束生成规则，帮助生成整数限制。如下列代码所示：

| C Code | interpretation |
|---|---|
| char s[n]; | $n^{\infty} \subseteq$ alloc(s) |
| strlen(s); | len(s) – 1 |
| strcpy(dst, src); | len(src) $\subseteq$ len(dst) |
| strncpy(dst, scr, n); | min(len(src), n) $\subseteq$ len(dst) |
| s = “foo”; | 4 $\subseteq$ len(s), 4 $\subseteq$ alloc(s) |
| p = malloc(n); | n $\subseteq$ alloc(p) |
| strcat(s, suffix); | len(s) + len(suffix) – 1 $\subseteq$ len(s) |
| p = getenv(…); | $[1, \infty] \subseteq$ len(p), $[1, \infty] \subseteq$ alloc(p), |
| gets(s); | $[1, \infty] \subseteq$ len(s) |
| sprintf(dst, “%s” src); | len(src) $\subseteq$ len(dst) |
| p[n] = ‘\0’; | min(len(p), n+1) $\subseteq$ len(p) |
| p = strchr(s, c) | p = s + n, [0, len(s)] $\subseteq$ n |

通过上面的介绍，已经知道如何判断缓冲区的大小和起始地址，我们以缓冲区起始地址来标示缓冲区，这样就可以很容易得到限制条件。

另一方面，由于基于 Linux 的二进制文件汇编代码中采用 call 指令来调用库函数，而且 IDA 可以帮助分析出所调用的标准库函数名，因此通过匹配定位这些库函数名，然后分析相应的压栈参数，可以方便地将 C 语言到限制规则的翻译转换为从汇编到限制规则的翻译。这样可以将上述基于 C 语言的整数限制影射规则，生成相应的整数限制条件。

(3)整数约束分析

在生成整数限制条件以后，我们开始进行实际的缓冲区溢出漏洞分析。虽然在整数限制生成模块中给出了每个变量的约束条件，但是对变量之间的关系一无所知。在整数约束分析模块中就尝试用一种图论中的概念解决这个问题。

如果 2 个变量 $v_i$与$v_j$ 之间有如 $f(v_i) \subseteq v_j$ 的约束关系，那么就构造如图 3 的约束关系。

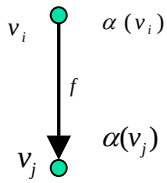


**图 3 约束关系**

分析的过程就是根据函数依赖关系图 FDG，从 main 函数开始以深度优先遍历整个函数关系树，如果存在一条整数限制就向图中加入一条如上图的边。如果 $f(\alpha(v_i)) \subset \alpha(v_j)$，那么就将这条边称为“活动(active)”的，只要找出所有这样活动的边就行了，这样所有不符合约束的语句，以及它们之间的关系就被找出来了。

### 3.3 动态分析模块

动态分析模块主要是进一步验证由静态分析所找到的缓冲区溢出漏洞。在动态分析中，主要采用错误注入和堆栈比较的方法。其基本思想是利用 ptrace 和 pstack 等系统命令对程序执行过程进行动态跟踪和记录。

在动态测试中，首先以正常状态执行被测试程序，记录下静态分析中存在缓冲区溢出漏洞的函数运行前后的堆栈的映像。然后，使用类似 FIST 的工具在疑似漏洞处注入错误的字符串，诱导程序溢出，并记录堆栈情况，从而能够进一步分析和确认缓冲区溢出漏洞。

通过动态分析，可以剔除在静态分析中发生的误判，提高整个分析的准确率。动态分析结束后，动态分析模块将生成最终的缓冲区溢出漏洞分析报表，记录通过该漏洞挖掘模型所分析的漏洞及其位置，方便作进一步的分析。

## 4 程序最终执行结果和对比分析

我们首先利用 Wagner 方法的工具测试了 sendmail8.7.5 的源程序。

该工具将检测到的错误分为 3 类：确定的漏洞，可能的漏洞，已经不确定的潜在漏洞，检测的结果是不存在肯定的漏洞，而可能的漏洞有 85 个，其中可以找到 3 个是 sendmail8.7.5 的官方漏洞。

接着，我们用自己开发的工具检测 sendmail8.7.5 的可执行文件，也不存在确定的漏洞，可能的漏洞有 106 个，其中可以找到 4 个官方漏洞。不过潜在漏洞数量多于 Wagner 工具检测的结果。

可以看出，我们的工具在发现漏洞的绝对数量上并不比基于源代码的工具数量少，而误报率仍然很高，我们的挖掘模型仍然只能是一种漏洞挖掘的辅助工具，而且我们的模型无法判断查找出的漏洞是否为可利用的漏洞，这些工作仍然需要安全工程师人工的分析。

## 5 存在的问题和主要解决途径

我们提出的基于二进制代码缓冲区溢出漏洞分析模型能够有效地发现存在于 Linux 的 ELF 文件中的缓冲区溢出漏洞，但是该模型仍然存在一些问题：

首先，该模型的分析结果会存在较多误报的缓冲区溢出漏洞，这主要是由于在静态分析中，无法准确地判断缓冲区运行过程中实际大小，只能估算缓冲区大小，导致误报和漏报缓冲区溢出漏洞。另一方面，由于动态分析过程中难以准确重现缓冲区溢出攻击发生的情况，不能有效地去除静态分析中的误报漏洞，因此，本模型仍然存在较多的误报和漏报。在以后的工作中，可以改进静态分析中对数据流和控制流的分析，使之更加准确，能够改善误报的问题。结合静态分析的结果，利用一些现有的错误注入算法，构造更为有效的错误输入来重现缓冲区溢出攻击，也可能改进漏洞分析结果。

其次，该模型只是用于判定缓冲区是否存在潜在的漏洞，而不能给出如何利用该漏洞进行攻击的方法，而且一些漏洞可能是不能利用或者难以利用的。对于这个问题，利用专家系统或者人工智能等方法实现漏洞利用的分析也许会得到比较令人满意的结果。

**致谢** 本文的直接动力与思想来自于上海交通大学计算机系密码与信息安全实验室的项目。在此，第一作者感谢郑东老师对本人长期以来的关心和指导，感谢郑燕飞老师、李晖以及实验室其他同学的各方面的指导和帮助！

### 参考文献

1 Wagner D, Foster J, Brewer E, et al. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities[C]. Proc. of Symposium on Network and Distributed Systems Security, San Diego, CA, 2000-02: 3-17.

2 Dor N, Rodeh M, Sagiv M. Cleanness Checking of String Manipulations in C Programs via Integer Analysis[C]. Proceedings of the 8th International Static Analysis Symposium, 2001-06.

3 Cowan C, Wagle P, Pu C. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade[C]. Proceedings of the DARPA Information Survivability Conference and Expo, 1999.

4 Haugh E. Testing C Programs for Buffer Overflow Vulnerabilities[D]. University of California at Davis, 2002.

5 Wong P. A Watermark for Image Integrity and Ownership Verification [C]. Proceedings of the IS & T PICS' 99, 1999: 374-379.