Fuzzing 技术综述 *

吴志勇, 王红川, 孙乐昌, 潘祖烈, 刘京菊
(解放军电子工程学院 网络工程系, 合肥 230037)

摘 要: 通过分析比较多种 Fuzzing 技术的定义,结合其当前发展所基于的知识和采用的方法,给出了 Fuzzing 技术的一个新的定义;重点从与黑盒测试技术的区别、测试对象、架构和测试数据产生机理四个方面总结了当前 Fuzzing 技术采用的一些新思想、新方法以及它们的缺陷。针对这些缺陷和实际应用中的需求,分别提出了当前 Fuzzing 技术下一步的具体研究方向和对应的研究方法。

关键词: Fuzzing 技术; 黑盒测试; 架构; 测试数据; 生成; 变异; 动态测试; 知识
中图分类号: TP391 **文献标志码**: A **文章编号**: 1001-3695(2010)03-0829-04
doi:10.3969/j.issn.1001-3695.2010.03.006

Survey on Fuzzing

WU Zhi-yong, WANG Hong-chuan, SUN Le-chang, PAN Zu-lie, LIU Jing-ju
(*Dept. of Network Engineering, Electronic Engineering Institute of PLA, Hefei* 230037, *China*)

Abstract: By analyzing and comparing several definitions of Fuzzing, this paper gave a new definition accroding to the knowledge and methods using currently, summerized its new ideas, new methods and corresponding defeats from these aspects like differences from black-box testing, framework and test data generation mechanism. Based on these defeats and the requiments from practical application, proposed concrete research directions and methods.

Key words: Fuzzing; black-box testing; construction; test data; generation; mutation; dynamic test; knowledge

Fuzzing 技术源于软件测试中的黑盒测试技术,它的基本思想是把一组随机数据作为程序的输入,并监视程序运行过程中的任何异常,通过记录导致异常的输入数据进一步定位软件中缺陷的位置。1990 年 Miller 等人[1]发现,通过简单的 Fuzzing 可以使运行于 UNIX 系统上的多于 25% 的程序崩溃;2002 年 Aitel[2,3]通过自己设计实现的 Fuzzing 工具 SPIKE 成功地发现了多个未知漏洞;2008 年 Godefroid 等人[4,5]利用 Fuzzing 工具 SAGE 发现大型 Windows 应用程序中二十多个未知漏洞。关于 Fuzzing 的定义,人们从不同的角度进行了描述。Miller 等人[1,6]认为 Fuzzing(或 Fuzz testing)是软件测试中的随机测试技术;Sutton 等人[7]认为 Fuzzing 的重要组成部分是暴力测试;Andrea 等人[8]认为 Fuzzing 是一种简单的黑盒测试技术;Oehlert[9]认为 Fuzzing 是黑盒测试技术中的边界测试技术,Vuagnoux[10]认为 Fuzzing 是黑盒测试、错误注入、压力测试的技术融合。总体来说,传统的 Fuzzing 是指一种非常简单的黑盒测试技术或者随机测试技术[1,6,7,11],用来发现软件的缺陷(flaws)。笔者认为 Fuzzing 是把随机测试和边界测试技术、协议和软件知识、具体执行和试探性攻击结合在一起的技术方法。

Fuzzing 作为一项发现软件错误的测试技术具备如下优点:其测试目标是二进制可执行代码,比基于源代码的白盒测试方法适用范围更广泛;Fuzzing 是动态实际执行的,不存在静态分析技术[12,13]中存在的大量的误报问题;Fuzzing 原理简单,没有大量的理论推导和公式计算,不存在符号执行技术[5,8,14~22]中的路径状态爆炸问题;Fuzzing 自动化程度高,无须在逆向工程过程中大量的人工参与。因此,Fuzzing 技术是一种有效且代价低的方法,在许多领域受到欢迎,许多公司和组织用其来提高软件质量[23~25],漏洞分析者使用它发现和报告漏洞[26],黑客使用它发现并秘密利用漏洞。国内对 Fuzzing 技术有了初步的研究[27]和简单的应用[28~31]。

1 早期 Fuzzing 技术

早期的 Fuzzing 技术仅是一种简单的随机测试技术[1,6],但却有效地发现了许多程序中的错误。文献[6]中设计的 Fuzzing 工具包括一个随机字符串产生器,产生包含可打印字符和控制字符的字符串;一些脚本用来加强测试过程的自动化,并记录程序崩溃的现场以协助追踪错误原因和地址。早期的程序中出现的错误也比较简单,如代码中因没有对输入的字符串的长度进行检查,而导致栈溢出。

由于早期的 Fuzzing 技术操作起来简单,与其测试目标程序的关联性小,它的优点包括:a)可用性,不需要获得目标程序的源代码就可以测试;b)复用性,如测试 FTP(file transfer protocol)的 Fuzzing 程序可以用来测试任何 FTP 服务器;c)简单性,无须过多了解目标程序。然而,Fuzzing 技术不可避免地带有随机测试产生大量冗余测试输入、覆盖率低导致发现软件缺陷概率低的缺点[32],同时带有黑盒测试的低智能性的缺点,

收稿日期: 2009-07-15; **修回日期**: 2009-08-30 **基金项目**: 电子工程学院博士生创新基金资助项目

作者简介:吴志勇(1982-),男,江苏大丰人,博士研究生,主要研究方向为网络安全(wuzhiyong0127@gmail.com);王红川(1983-),男,硕士研究生,主要研究方向为网络安全;孙乐昌(1951-),教授,河北昌邑人,主要研究方向为网络安全;潘祖烈(1976-),男,博士,主要研究方向为网络安全;刘京菊(1974-),女,讲师,硕士,主要研究方向为网络安全.

即黑盒测试只测试了程序的初始状态,而很多程序尤其是网络协议程序的很多错误是隐藏在程序的后序状态中的。

2　Fuzzing 技术的研究现状

2.1　与黑盒测试技术的区别

经过近二十年的发展,源于黑盒测试技术的 Fuzzing 技术显示出与前者的不同,主要表现在:a)测试需求的着眼点不同。Fuzzing 测试着眼于发现软件安全性相关的错误,黑盒测试着眼于测试软件的功能的正确性。b)测试用例的侧重点不同。由于测试需求的不同,Fuzzing 的测试用例大多数都是畸形的测试用例,黑盒测试的用例的大多数都是正确的测试用例。Fuzzing 为了提高测试用例的有效性,则必须提高测试用例的正确性,从而使测试用例的畸形数据能够达到程序的潜在不安全点。c)由于测试用例的产生机理不同,Fuzzing 为了产生有效的畸形数据,需要考虑到测试用例的数据格式、目标程序的结构流程和程序运行的中间状态;而黑盒测试只关心目标程序的外部接口和外部输入,从这个意义上讲,现在的 Fuzzing 技术更接近于灰盒测试。

2.2　Fuzzing 的测试对象

在 Fuzzing 技术的有效性进一步得到验证之后,出现了许多针对特定类型应用程序或者协议的 Fuzzing 工具,如针对浏览器的 mangleme、针对文件应用程序的 FileFuzzing 和 SPIKEfile、针对 ActiveX 的 COMRaider 和 AxMan,尤其突出的是 2002 年出现的针对网络协议的 SPIKE[2,3]。后来,进一步出现了针对性更强、功能也更为单一的 Fuzzing 工具,如针对 IRC 协议的 ircfuzz[33],针对 DHCP 协议的 dhcpfuzz[34] 和针对 FTP 的 tftpfuzz[34] 工具等。当前的 Fuzz 工具主要是针对文件格式应用程序和网络协议应用程序,但也出现了可以测试浏览器[7]、操作系统内核[35]、Web 应用程序[36] 的 Fuzz 工具。另外,引入了 agent思想的工具 Peach[35] 可以对分布式系统进行 Fuzz 测试。

2.3　Fuzzing 的架构

最为简单的 Fuzzing 架构如图 1 所示,包含引擎和监视两个模块。引擎模块的功能是产生 Fuzzing 需要的数据并把数据发送到目标程序使之运行;监视模块的功能是监视目标程序的运行状态是否出错。早期 Fuzzing[1,6] 的实现就是采用了该模型,监视模块功能的实现借助于简单的脚本来记录程序出错的信息。

图1　简单Fuzzing架构

随着 Fuzzing 技术的进一步发展,出现了如图 2 所示的架构设计。为了避免产生大量无效[9] 的测试数据,新架构中出现了参数脚本和样本文件。参数脚本给出了引擎生成的测试用例中的数据的格式、长度等与数据之间的一些关系,如 SPIKE、Sulley[37] 使用的类 C 格式的脚本,Peach[38] 使用的 XML 格式的脚本。样本文件是许多基于变异技术(mutation-based)的数据生成方式的 Fuzzing 工具用来变异测试数据的基准。基于样本文件产生的测试数据可以大大提高测试用例的有效性,可以提高测试的代码覆盖率,可以减轻测试用例构造的复杂度。Fuzzing 测试文件格式应用程序依据的样本文件是其相应格式(如 DOC、PDF 等)的样本文件(如 Peach),Fuzzing 测试网络协议应用程序的样本文件是通过嗅探工具(如 Ethereal[39] 或 WireShark[40])的数据包转换的样本文件(如 Peach、Sulley、AutoDafe[41]、SPIKE Proxy[36] 等)。

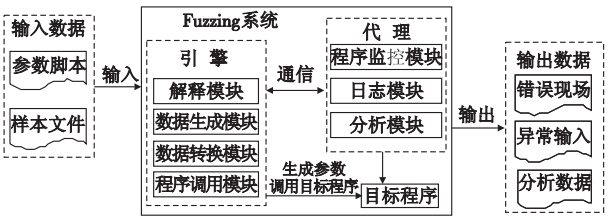

图2　当前Fuzzing架构

在图 2 所示的架构中,Fuzzing 引擎的模块划分粒度更为细化,其目的是加强代码的可复用性和整个 Fuzzing 构架的灵活性,以方便用户根据需求快速制定适合其他多种协议的 Fuzzing 程序。

Fuzzing 的监视模块发生了很大的变化,转换为功能更为丰富的代理(agent)模块(如工具 Peach、Sulley),一方面可以并行 Fuzzing 的过程以提高 Fuzzing 的效率,一方面可以使引擎和代理分离开来在不同的机子上运行,可以对分布式应用程序进行 Fuzzing 测试。a)许多代理都包括程序监控子模块,用来监视和控制程序的运行情况。由于开发调试器的工作量大,大多数工具使用的是第三方的调试工具,如 Peach,有的工具使用的是可以定制特殊需求的调试器,如 Sulley、AutoDafe。b)代理中还添加了日志模块,主要用来记录发生异常的现场,以协助用户进一步定位应用程序发生错误的位置,还要记录使之发生异常的测试用例或者程序输入,以使该错误可以恢复。另外,极少数工具如 Sulley 添加了分析模块,用来统计 Fuzzing 测试的结果信息,如代码覆盖率等。

2.4　Fuzzing 测试数据的产生机理

早期的 Fuzzing 技术就是随机测试技术,测试数据多数是随机产生的畸形数据。为了提高 Fuzzing 产生数据的有效性,出现了下面两种产生测试数据的思想:

a)基于格式分析和程序理解相结合的数据产生方法,其代表工具有 SPIKE、Peach、Sulley、AutoDafe 等。通过对文件和协议的理解,该方法产生的数据可以有效地越过应用程序中对固定字段、校验和、长度的检查,从而使 Fuzzing 的测试数据的有效性大大加强。该方法又可以分为基于生成技术的方法(generation-based)和基于变异技术的方法(mutation-based)[42] 或两者相结合(如工具 Peach)的方法。基于生成技术的测试数据产生方法通常是给出文件格式或者网络协议具体的描述规则,然后依据此规则产生测试数据。该方法需要用户对格式或者协议有非常深的了解,并需要大量的人工参与[10]。基于变异技术的数据产生方式通常是在对格式或者协议有所了解的前提下,对获得的样本数据中的某些域进行变化,从而产生新的变异数据。该方法对初始值有着很强的依赖性,不同的初始值会带来差异很大的代码覆盖率,从而会产生差异很大的 Fuzzing 效果[42]。还有一种完全自动化的基于变异技术的数据产生方式,如工具 AutoDafe 和 SPIKE Proxy,该方法利用协议的自动分析技术[7,36,41] 实施对测试数据的自动生成。但由于协议自动分析技术的准确率还有待进一步提高,其 Fuzzing 测

试效果并不理想。总之,基于格式分析和程序理解相结合的数据产生方法的优点是执行效率比较高、应用范围广、通用性强,缺点是仍然需要大量的人工参与来进行多种知识(如协议知识、数据格式知识、应用程序知识)的获取并实现这些知识到测试用例的转换。

b)基于静态分析与动态测试相结合的数据产生方法。通过与静态分析技术、符号执行技术、具体执行技术等多种技术[4,5,8]相结合,从而在达到一个较高的代码覆盖率的测试基础上进行 Fuzzing 测试。本质上来说,这种方法是白盒测试与 Fuzzing 测试技术的融合。该方法通过借助软件测试中的技术使 Fuzzing 技术得到一个不错的代码覆盖率,该方法的缺点是仍然无法克服符号执行中的状态爆炸问题,也无法完全自动解决部分程序自带的高强度的程序检查(如校验和和加密数据)问题;另外,该方法采用了类似于穷搜索的思路,每次执行需要大量的时间,效率低;而且每次执行都需要复杂的符号运算,从而消耗了大量的时间。

3　Fuzzing 技术的研究方向

Fuzzing 技术以其简单有效的优点得到了越来越多的关注,结合上述技术的不足和实际应用中的需求,其研究方向可以归纳为以下几个方面:

a)Fuzzing 测试平台的通用性研究。由于 Fuzzing 的测试对象越来越广泛,如何构建通用的、可扩展性强的通用平台对于提高 Fuzzing 技术的整体发展十分必要。通用的测试平台应该具备下面几个功能:具备数据格式的解释功能从而产生适合多种数据格式的有效的畸形测试数据;具备独立的、可定制的数据产生变异功能,可以产生多种类型的、针对性强的畸形数据;具备可操作的跟踪调试功能以反馈运行时的多种信息;具备高效的引擎以协调多个模块之间的自动化运行。

b)知识获取的自动化程度的提高。实际的 Fuzzing 测试过程显示绝大部分的时间都花费在输入数据格式、程序状态转换的人工分析上;提高测试数据或通信协议的自动化分析或半自动化分析水平可以有效提高 Fuzzing 的测试效率。

c)多维的 Fuzzing 测试用例生成技术研究。当前的Fuzzing 测试用例生成技术都是一维的,即每次只变化一个输入元素,而许多漏洞是由多个输入元素共同作用引起的。多维测试用例生成技术的研究可以有效扩展 Fuzzing 发现的漏洞范围,但是多维 Fuzzing 测试用例会带来类似于组合测试中的状态爆炸问题,现有的组合测试理论成果对于解决 Fuzzing 多维测试中的状态爆炸问题有一定的借鉴意义。

d)智能的测试用例生成技术研究。利用漏洞知识给出合适的导向,结合智能算法像遗传算法[10]、模拟退火算法可以有效避免 Fuzzing 随机性强、漏洞漏报率高的缺点。

e)Fuzzing 测试的程序多状态的自动覆盖技术研究。可以解决需要人工参与才能覆盖程序的多个状态问题,从而提高整体 Fuzzing 测试的效率。

f)Fuzzing 测试效果的评估技术研究。以黑盒测试中的代码覆盖率来评价 Fuzzing 的测试效果是不直接、不科学的;从覆盖不安全代码的覆盖率、程序状态的覆盖率、输入边界测试的充分性、缓冲区边界覆盖的充分性、测试数据的有效性和知识获取的充分性等多个角度来衡量 Fuzzing 的测试效果会更加科学,也会更好地指导 Fuzzing 测试用例的生成和 Fuzzing 技术的进一步发展。

参考文献:

[1] MILLER B P, FREDRIKSON L, SO B. An empirical study of the reliablity of UNIX utilities[J]. Communications of the ACM, 1990, 33(2):32.

[2] AITEL D. The advantages of block-based protocol analysis for security testing[R]. New York: Immunity Inc, 2002.

[3] SPIKE[EB/OL]. (2009-06). http://www.immunitysec.com/resources-freesoftware.shtml.

[4] GODEFROID P, LEVIN M, MOLNAR D. Active property checking[C]//Proc of the 8th ACM International Conference on Embedeling Software. 2008:19-24.

[5] GODEFROID P, LEVIN M, MOLNAR D. Automated whitebox fuzz testing[C]//Proc of Network Distributed Security Symposium. 2008.

[6] MILLER B P, KOSKI D, LEE C P, *et al*. Fuzzing revisted: a reexamination of the reliability of UNIX utilities and services[R]. Madison: University of Wisconsin Madison, 1995.

[7] SUTTON M, GREENE A, AMINI P. Fuzzing: brute vulnerability discovery[M]. [S.l.]: Pearson Education Inc, 2007: 16.

[8] ANDREA L, LORENZO M, MATTIA M, *et al*. A smart fuzzer for x86 executables[C]//Proc of the 3rd International Workshop on Software Engineering for Secure Systems. [S.l.]: IEEE Computer Society, 2007:7.

[9] OEHLERT P. Violating assumption with fuzzing[J]. IEEE Security and Privacy, 2005, 3(2):58-62.

[10] VUAGNOUX M. Autodafe: an act of software torture[EB/OL]. (2006). http://autodafe.sourceforge.net/docs/autodafe.pdf.

[11] FORRESTER J E, MILLER B P. An empirical study of the robustness of windows NT applications using random testing[C]//Proc of the 4th USENIX Windows System Symposium. 2000:6.

[12] LAROCHELLE D, EVANS D. Statically detecting likely buffer overflow vulnerabilities[C]//Proc of the 10th on USENIX Security Symposium. Berkeley: USENIX Association, 2001.

[13] DOR N, RODEH M, SAGIV M. CSSV: towards a realistic tool for statically detecting all buffer overflows in C[J]. ACM SIGPLAN Notices, 2003, 38:155-167.

[14] BUSH W R, PINCUS J D, SIELAFF D J. A static analyzer for finding dynamic programming errors[J]. Software-Practice Experience, 2000, 30(7):775-802.

[15] XIE Yi-chen, CHOU A, ENGLER D. ARCHER: using symbolic, path-sensitive analysis to detect memory access errors[C]//Proc of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2003:327-336.

[16] GODEFROID P, KLARLUND N, SEN K. DART: directed automated random testing[J]. ACM Sigplan Notices, 2005, 40(6):213-223.

[17] CADAR C, GANESH V, PAWLOWSKI P M, *et al*. EXE: automatically generating inputs of death[J]. ACM Trans on Inf System Secur, 2008, 22:1-38.

[18] CRISTIAN C, VIJAY G, PETER M P, *et al*. EXE: automatically generating inputs of death[C]//Proc of the 13th ACM Conference on Computer and Communications Security. New York: ACM Press, 2006:322-335.

[19] YANG Jun-feng, SAR C, TWOHEY P, *et al*. Automatically generation malicious disks using symbolic execution [C]//Proc of IEEE Symposium on Security and Privacy. [S. l.]: IEEE Computer Society, 2006.

[20] GODEFROID P. Compositional dynamic test generation [C]//Proc of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM Press, 2007:47-54.

[21] CADAR C, DUMBAR D, ENGLER D. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs [C]//Proc of the 8th USENIX Symposium on Operating System Design and Implementation. 2008.

[22] ANAND S, GODEFROID P, TILLMANN N. Demand-driven compositional symbolic execution, MSR-TR-2008-138 [R]. [S. l.]: Microsoft Research, 2007:367-381.

[23] HOWARD M. Inside the windows security push [J]. IEEE Security & Privacy, 2003,1(1):57-61.

[24] LIPNER S, HOWARD M. The trustworthy computing security development lifecycle [EB/OL]. (2005). http://msdn. microsoft. com/security/default. aspx? pull = /library/en-us/dnsecure/html/sdl. asp.

[25] MAXWELL S A. The bulletproof penguin, secure linux through fuzzing [EB/OL]. (2001). http://home. pacbell. net/s-max/scott/bulletproof-penguin. html.

[26] GRIMES R. The buzz about fuzzers [EB/OL]. (2005). http://www. infoworld. com/article/05/09/09/37OPsecadvise_1. html.

[27] 邵林,张小松,苏恩标. 一种基于 Fuzzing 技术的漏洞发掘新思路[J]. 计算机应用研究,2009,26(3):1086-1088.

[28] 吴毓书,周安民,吴少华,等. 基于 Fuzzing 的 ActiveX 控件漏洞发掘技术[J]. 计算机应用,2008,28(9):2252-2254.

[29] 魏瑜豪,张玉清. 基于 Fuzzing 的 MP3 播放软件漏洞发掘技术[J]. 计算机工程,2007,33(24):158-160,167.

[30] 成厚富,张玉清. 基于 Fuzzing 的蓝牙 OBEX 漏洞挖掘技术[J]. 计算机工程,2008,34(19):151-153,156.

[31] 高峻,徐志大,李健. 针对符合文档的 Fuzzing 测试技术[J]. 计算机与数字工程,2008,36(12):116-119.

[32] OFFUTT A J, HAYES J H. A semantic model of program faults [C]//Porc of ACM SIGSOFT International Symposium on Software Testing and Analysis. San Diego: ACM Press, 1996.

[33] Ircfuzz [EB/OL]. (2009-06). http://www. digitaldwarf. be/products/ircfuzz. c.

[34] Dhcpfuzz [EB/OL]. (2009-06). http://www. digitaldwarf. be/products/dhcpfuzz. pl.

[35] Tftpfuzz [EB/OL]. (2009-06). http://www. digitaldwarf. be/products/tftpfuzz. py.

[36] SPIKE Proxy [EB/OL]. (2009-06). http://www. immunitysec. com/resources-freesoftware. shtml.

[37] Sulley [EB/OL]. (2009-06). http://www. fuzzing. org.

[38] Peach [EB/OL]. (2009-06). http://www. peachFuzzer. com; http://peachfuzz. sourceforge. net. http://peachfuzz@ googlegroups. com.

[39] Ethereal [EB/OL]. (2009-06). http://www. ethereal. com.

[40] Wireshark [EB/OL]. (2009-06). http://www. wireshark. org.

[41] AutoDafe [EB/OL]. (2009-06). http://autodafe. sourceforge. net; http://autodafe. sourceforge. net/docs/autodafe. pdf.

[42] MILLER C, PETERSON Z N J. Analysis of mutation and generation-based fuzzing [EB/OL]. (2009-07). http://securityevaluators. com/files/papers/analysisfuzzing. pdf.

[43] LIU Guang-hong, WU Gang, ZHENG Tao, *et al*. Vulnerability analysis for x86 executables using genetic algorithm and fuzzing [C]//Proc of the 3rd International Conference on Convergence Hybird Information Technology. 2008:491-497.

[44] KAKSONEN R. A functional method for assessing protocol implementation security [D]. Oulu, Finland: University of Oulu, 2001.

[45] SUTTON M, GREENE A. The art of file format fuzzing [EB/OL]. (2004). http://www. blackhat. com/presentations/bh-Jp-05. bh_jp-05-sutton-greene. pdf.

[46] GPF [EB/OL]. (2009-06). http://www. appliedsec. com/resources. html.

(上接第 815 页)

[26] PARK K M, KIM S H, LEE D G. Boosting lexical knowledge for biomedical named entity recognition [C]//Proc of Joint Conference on Natural Language Processing in Biomedicine and It's Applications. 2004:76-79.

[27] LEE K J, HWANG Y S, KIM S, *et al*. Biomedical named entity recognition using two-phase model based on SVMs [J]. Journal of Biomedical Informatics, 2004,37(6):436-447.

[28] ZHAO Shao-jun. Named entity recognition in biomedical texts using an HMM [C]//Proc of Joint Conference on Natural Language Processing in Biomedicine and It's Applications. 2004:87-90.

[29] BORTHWICK A E. A maximum entropy approach to named entity recognition [D]. New York: New York University, 1999.

[30] LAFFERTY J, McCALLUM A, PEREIRA F. Conditional random fields probabilistic models for segmenting and labeling sequence data [C]//Proc of the 18th International Conference on Machine Learning. San Francisco: Morgan Kaufmann Publisher, 2001:282-289.

[31] FINKEL J, DINGARE S, MANNING C D, *et al*. Exploring the boundaries: gene and protein identification in biomedical text [J]. BMC Bioinformatics, 2005,6(1): S5.

[32] DINGARE S, NISSIM M, FINKEL J, *et al*. A system for identifying named entities in biomedical text: how results from two evaluations reflect on both the system and the evaluations [J]. Comparative and Functional Genomics, 2005,6(1-2): 77-85.

[33] SCHWARTZ A S, HEARST M A. A simple algorithm for identifying abbreviation definitions in biomedical text [C]//Proc of Pacific Symposium on Biocomputing. 2003:451-462.

[34] YOSHIDA K, TSUJII J. Reranking for biomedical named-entity recognition [C]//Proc of Human Language Technology Conference. Morristown: Association for Computational Linguistics, 2007:215-222.

[35] WANG Hao-chang, ZHAO Tie-jun, TAN Hong-ye, *et al*. Biomedical named entity recognition based on classifiers ensemble [J]. International Journal of Computer Science and Applications, 2006,5(2):1-11.

[36] CHAN S, LAM W, YU Xiao-feng. A cascaded approach to biomedical named entity recognition using a unified model [C]//Proc of the 17th International Conference on Data Mining. Washington DC: IEEE Computer Society, 2007:93-102.

[37] VLACHOS A, GASPERIN C. Bootstrapping and evaluating named entity recognition in the biomedical domain [C]//Proc of Workshop on Linking Natural Language Processing and Biology. Morristown: Association for Computational Linguistics, 2006:138-145.

[38] GU B, POPOWICH F, DAHL V. Recognizing biomedical named entities in Chinese research abstracts [M]//Advances in Artificial Intelligence. Berlin/Heidelberg: Springer, 2008:114-125.