模糊测试技术研究综述

张 雄 李舟军
(北京航空航天大学计算机学院 北京 100191)

摘 要 软件中的安全漏洞可能导致非常严重的后果,因此漏洞挖掘已成为网络与信息安全领域的重大课题和研究热点。目前常用的漏洞挖掘技术包括静态分析、动态分析、二进制比对、模糊测试等。随着软件的规模和复杂度不断增大,模糊测试具有其它漏洞挖掘技术无法比拟的优势。首先介绍和分析了各种漏洞挖掘技术的优点和缺点;然后分别详细描述了模糊测试的研究进展、模糊测试的过程、测试用例的生成技术;最后介绍了模糊测试在各个领域的应用,并对其发展方向进行了展望。

关键词 软件安全,漏洞挖掘,模糊测试,测试用例生成
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.5.001

Survey of Fuzz Testing Technology

ZHANG Xiong LI Zhou-jun
(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract Security vulnerabilities in software may lead to serious consequences, and vulnerability exploiting has become a hot area of research in network and information security. Popular vulnerability exploiting technologies include static analysis, dynamic analysis, binary code comparison, fuzz testing and so on. Along with the expansion of the scale and complexity of software, fuzz testing has incomparable advantages which other vulnerability exploiting technology can't provide. Firstly, both advantages and disadvantages of various vulnerability exploiting technology are discussed. Secondly, an account of the research advances of fuzz testing the procedure of fuzz testing and test case generation technology were described in detail. Finally, the applications of fuzz testing were shown and the trend of future study was discussed.

Keywords Software security, Vulnerability exploiting, Fuzz testing, Test case generation

1 引言

随着计算机的广泛使用和计算机网络的大范围普及,软件产业得到了飞速发展,并且已从国防技术、工业控制、航空航天、金融证券、邮电通信、医疗卫生等专业领域渗透到了人们的日常生活中。随着软件种类的增多,软件的功能越来越强,其规模也越来越大,软件的安全性问题日益凸显。

近年来,利用软件漏洞进行恶意攻击的行为层出不穷。根据国家互联网应急中心的报道[1]可知,2002年到2013年期间,每年都会新增大量的漏洞报告。一些不法分子可以利用这些漏洞在极短时间内攻击用户电脑、服务器等,获取用户的隐私信息或者篡改用户存储在网络空间中的私密数据。最近发生的棱镜事件、OpenSSL相关的HeartBleed漏洞、携程网泄密事件、12306密码泄露事件等,无疑将软件安全问题推到了风口浪尖。

漏洞挖掘已成为软件安全领域的一个重要研究方向。目前常用的漏洞挖掘技术分为静态分析、动态分析、二进制比对、模糊测试等[2]。随着软件规模的爆炸式增长,除模糊测试以外的其他几种漏洞挖掘技术,或者存在状态爆炸、路径爆炸、约束求解困难、耗时长等问题,或者存在误报率高等缺点。相比这些方法,模糊测试具有自动化程度高、系统消耗低、误报率低、不依赖于目标程序源代码等优点,正是这些优点激起了人们对模糊测试研究的热情。

2 概述

2.1 常用漏洞挖掘技术

目前常用的漏洞挖掘技术分为静态分析、动态分析、二进制比对、模糊测试等。

静态分析是指在不运行软件的前提下进行的一种分析技术。其主要通过对目标程序的词法、语法、语义分析来发现软件中潜在的安全漏洞[2],特别是函数调用中参数的来源以及函数返回值,通过对参数进行来源分析与跟踪,很容易发现其中未对参数进行边界检查而造成的缓冲区溢出、堆溢出等类型的安全漏洞。常用的静态分析工具根据分析对象可分为两类:1)针对有源代码的源码扫描工具,常用的有ITS4[3]、FLAWFINDER[4]等,主要用于检查不符合安全规范的函数

到稿日期:2015-05-13 返修日期:2015-07-04 本文受国家自然科学基金(61170189,61370126),国家863计划(2015AA016004),博士点基金(20111102130003)资助。

张 雄(1989—),男,硕士生,主要研究方向为网络与信息安全,E-mail:1025679612@qq.com;李舟军(1963—),男,教授,博士生导师,主要研究方向为数据挖掘与文本挖掘、网络与信息安全。

调用(如C中的strcpy、strcat)、指针等;2)对于二进制程序的反汇编扫描工具,bugscan和bugaudit是其中的代表,都是在通过IDA PRO[5]在目标程序反汇编的基础上进行分析[6]。静态分析易集成到开发过程中,自动化程度较高,但是缺少运行时的动态信息,无法对潜在的漏洞进行确认,误报率较高。

动态分析是一种通过动态加载并运行的目标软件,监测程序运行时的堆栈信息、内存使用情况、变量的值等状态信息以及程序的输出来验证或发现软件漏洞的技术[2,7]。与静态分析相比,动态分析需要运行时系统的支持与具体的输入数据,同时,还可能借助于OllyDbg[8]、WinDbg[9]、SoftICE[10]等调试工具。动态分析技术监测到的异常基本都是由程序漏洞引起的,对其进行分析即可发现相应的漏洞,因而,动态分析具有极高的准确率。然而,动态分析往往需要对目标软件进行逆向分析,涉及的技术较复杂,对分析人员的要求较高,自动化程度不高,有着较大的局限性。

二进制比对也称为补丁比对。作为软件安全漏洞的补救措施,软件开发商会定期或不定期地提供相应的修补程序即补丁。补丁对比技术通过对比添加补丁前后目标程序的变化来精确定位漏洞的位置和成因,有经验的人员结合数据流分析等方法能够在短时间内找出漏洞的成因并给出漏洞的利用代码。目前常用的补丁对比方法有二进制字节比对、二进制程序反汇编后的文本比对和结构化二进制比对等[2]。补丁作为高危漏洞的一种紧急补救措施,存在如下不足:1)补丁对目标程序作修改时,软件开发商往往仅考虑了漏洞的上下文,未必考虑到整个软件的系统环境,因此补丁对目标漏洞的修补未必一定完善;2)补丁程序通常是一种紧急采用的措施,补丁程序本身的安全性难以保证,补丁自身并没有经过严格的安全测试,虽然修补了原有的漏洞,但是补丁程序可能引入新的安全漏洞,更重要的是,二进制比对技术并不能用于挖掘新的漏洞,一般只是用于研究已经被发现的漏洞。

模糊测试是一种基于缺陷注入的自动化软件漏洞挖掘技术,其基本思想与黑盒测试类似。模糊测试通过向待测试的目标软件输入一些半随机的数据并执行程序,监控程序的运行状况,同时记录并进一步分析目标程序发生的异常来发现潜在的漏洞[11]。所谓半随机是指输入的数据对于目标程序来讲,其重要的数据格式(如指定文件格式的魔数、CRC校验)和大部分数据是有效且合法的,与此同时,输入的其它部分却是不满足目标程序输入数据格式规约的非法数据。由于目标程序在编写时未必考虑到对所有非法数据的出错处理,因此半随机数据很有可能造成目标程序崩溃,从而触发相应的安全漏洞。

2.2 模糊测试的优点和缺点

一次典型的模糊测试包含以下过程:确定测试目标、确定目标程序的预期输入、生成测试用例、执行测试用例、异常监视、异常分析与漏洞确认。在模糊测试的过程中,测试用例执行、异常监视这两个重要的过程完全可以自动化实现。而且,通过模糊测试技术发现的漏洞一般是真正存在的(原因是对半有效数据的处理不当),即模糊测试技术存在误报率低的优点。其它的漏洞挖掘方法往往需要对目标程序的源代码或二进制代码进行深入的分析[12],这是过程的开销巨大,而模糊测试并不需要对目标程序的源代码或二进制程序进行分析即可进行。

相比其它漏洞挖掘方法,模糊测试有着巨大的优势,与此同时,其不足也十分明显。目前模糊测试技术仍然存在许多局限性,模糊测试的一些缺点与局限性如下。

1)对访问控制漏洞无能为力

某些应用支持多级权限,每个用户都有相应的权限级别,不同级别的账户权限不同。例如,在教务管理系统中,存在管理员、学生、老师等角色,管理员拥有增加一门课程、删除学生某门课成绩、修改某学生的成绩等几乎所有权限;老师拥有修改某些课程所有选课学生成绩的权限;学生则只有查询自己所选的每门课分数的权限。在这类系统中,一个基本的权限控制就是要保证低权限用户不能执行高权限用户所具有的操作。

在模糊测试中,违反权限控制的安全漏洞很难被发现,比如,对于教务管理系统,数据库权限控制错误将导致学生可以修改自己的分数,然而,这类错误在模糊测试中是无法发现的,其根本原因是模糊测试系统无法理解程序的逻辑,它并不知道学生不能修改选课分数。事实上,即使可能修改代码让模糊测试系统实现这些逻辑,这样的模糊测试系统也无法在其它地方再次被重复使用[13]。

2)无法发现糟糕的设计逻辑

糟糕的逻辑往往并不会导致程序崩溃,而模糊测试发现漏洞的一个最重要的依据就是监测目标程序的崩溃,因此,模糊测试对这种类型的漏洞也无能为力。以下就是一个典型的例子。

在VERITAS Backup Exec for Windows Servers中存在一个允许攻击者远程控制Windows服务器并完全操纵注册表的漏洞,存在该漏洞的原因是Windows中某个实现远程过程调试(RPC)的接口在不需要认证的情况下就具有修改注册表的权限。这个漏洞根本上源自Windows系统设计上的一个失误,设计人员认为攻击者也许会花费更多的时间来破解IDL(接口定义语言),而不是去寻找RPC模块中的漏洞。虽然模糊测试可以发现目标程序在使用RPC时不进行参数检查从而导致的某些比较"低级"的错误,但是模糊测试无法确定这些错误是否真正会导致系统不安全。

3)无法识别多阶段安全漏洞

一次典型的攻击过程通常通过连续利用若干个安全漏洞来实现。首先利用漏洞使机器接受未授权的访问,然后再利用其它漏洞缺陷获得更大的权限,紧接着获得机器的完全控制权,最后达到恶意攻击的目的。模糊测试对识别单独的漏洞很有用,但通常而言,模糊测试对那些小的漏洞序列构成的高危漏洞却毫无办法。

4)无法识别多点触发漏洞

当前的模糊测试技术往往只能挖掘出由单个因素引起的漏洞,而对于需要多条件才能触发的漏洞却无能为力。近年来,相关学者试图将模糊测试用于挖掘多点触发漏洞,提出了多维模糊测试的概念并进行了相关的研究,然而,多维模糊测试面临着组合爆炸等亟待解决的难题,目前其研究进展缓慢。

3 模糊测试技术研究进展

模糊测试最早可追溯到1989年,威斯康辛大学的B. P. Miller开发了一个模糊测试工具并命名为Fuzz,用来对UNIX系统上的软件进行安全性测试。B. P. Miller等人发

现，通过向 UNIX 系统上的软件输入各种随机字符串，可使其中多于 25%的程序崩溃[11]。

1999 年，芬兰 Oulu 大学的协议测试项目组研发了网络协议安全测试软件 PROTOS[14]，并获得了微软公司的资金支持。PROTOS 使用了灰盒测试方法，首次提出了测试集的概念，即在分析协议归约的基础上，产生违背归约或者很可能无法正确处理的报文。不久之后，相关学者利用 PROTOS 发现了众多厂商 SNMP、SIP、LDAP 协议软件中诸如 CAN-2003-131 等严重的漏洞，最近网络出现的与 OpenSSL 相关的 HeatBleed 漏洞的发现工具就是在 PROTOS 的基础上发展起来的。PROTOS 是模糊测试技术发展过程中的一个重要的里程碑。

2002 年，Immunitysec 公司的 Dave Aitel 开发了一款通用协议测试框架工具 SPIKE[15,16]。SPIKE 主要用于测试基于网络的应用程序，采用了一种基于块的方法，能够有效地对包含变量和相应变量长度的数据块进行测试，同时，SPIKE 可以从大量的数据样例中自动学习目标软件的数据规约。作为第一个可以实现自定制的模糊测试器框架，SPIKE 的发布是模糊测试历史上一个最重要的里程碑。

2004 年，IOACTIVE 发布了著名的跨平台模糊测试框架 Peach[17]，它采用 Python 语言编写。Peach 几乎可以对所有常见的对象进行模糊测试，例如文件结构、COM 对象、网络协议、API 接口等。使用 Peach 时，其主要工作就是定义一个 XML 文件来引导测试过程。Peach 是一个灵活的模糊测试框架，最大程度地实现了代码重用。

2005 年，iDefense 公司开发了基于 Windows 平台的文件格式模糊测试工具 FileFuzz[13,18] 和基于 Linux 平台的 SpikeFile[13]，开启了文件类漏洞挖掘的热潮。随后几年，相当数量的类似漏洞相继出现，其中大多数漏洞被发现不久就开始危及到广泛使用这些文件格式的各种应用软件。由于文件格式越来越复杂，相应的漏洞数目越来越多，其危害相当严重，并且文件类漏洞的挖掘相对较容易，因此文件格式的安全漏洞挖掘最终成为模糊测试的主要应用领域之一。

2006 年左右，针对 ActiveX 控件的模糊测试开始兴起，David Zimmer 和 H. D. Moore 分别发布了测试工具 COMRaider[19] 和 AxMan[20]。这两款工具测试的对象是那些在微软 Internet Exploer 浏览器中能够被 Web 应用程序实例化的 ActiveX 控件，这两款软件发现了 ActiveX 控件中诸如 Heap Spray 之类的严重漏洞。此类应用程序的用户群体十分庞大且漏洞可被远程利用，因此 ActiveX 控件漏洞具有非常高的风险。正如之后大量测试结果证明的那样，ActiveX 控件是一种非常适合模糊测试的对象，因为它包括了调用接口、函数原型和实际的控制逻辑，所以十分便于对其实施自动化的模糊测试。

2007 年，密歇根州立大学的 Jared DeMott 提出了灰盒测试思想，并设计了更加高效的模糊测试器 EFS[21,22]。EFS 将代码覆盖与模糊测试技术相结合，通过反汇编工具找到应用程序中所有可能测试的函数，并逐一测试，挖掘了多个 Golden FTP、IIS FTP 和 SMTP 中的安全漏洞。

2008 年，微软研究院的 Patrice Godefroid[23,24] 在模糊测试方法中引入了白盒测试(White-box)思想，同时将符号执行和约束求解的方法应用于模糊测试中，使得模糊测试更加精确，覆盖率也更高。

2009 年，麻省理工学院的 Vijay Ganesh 提出了基于污点传播的白盒模糊测试技术，开发了一款模糊测试工具 BuzzFuzz[25]，并发现了开源软件 Swfdec 和 MuPDF 中多个著名的安全漏洞。

2010 年，北京大学的王铁磊将符号执行和细颗粒度动态污点传播技术相结合，提出了一种绕过校验和机制的方法，并开发了相应的工具 TaintScope[26,27]，发现了包括 Adobe Acrobat、Google Picasa、Microsoft Paint、ImageMagick 在内的多个未知漏洞。

2011 年之后，众多学者将遗传算法、启发式算法等用于生成模糊测试用例，大大提高了测试用例的生成效率[28,29]。同时，多维模糊测试研究的兴起也弥补了传统模糊测试只能挖掘单点触发漏洞的不足。

4 模糊测试过程

模糊测试方法多种多样，没有一种对于所有软件都适合的方法，决定使用哪种模糊测试方法完全依赖于被测的程序、测试者的技术水平以及目标程序所接受的输入数据格式。但是，无论采用什么样的模糊测试方法对何种目标程序进行模糊测试，其测试过程都应包含如图 1 所示的几个相同的基本阶段[13]。

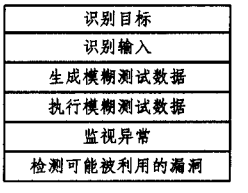

图 1 模糊测试的几个阶段

4.1 确定测试目标

确定测试目标是整个模糊测试过程中的第一步，有了明确的目标，才能决定使用的测试工具和测试方法。在此过程中，可以借鉴风险模型的测试方法，根据被测的程序，在一些典型的漏洞收集网站如 SecurityFocus[30]、Secunia[31]、CNVD[32] 等查找软件开发商历史上曾出现的安全漏洞，分析这些漏洞的形成原因及编码习惯，有针对性地选择相应的模糊测试工具和方法。同时对于类似 Windows 上的 DLL、Linux 上的 SO 等类型的被多个程序使用的共享库，也应该成为重点的测试对象，因为在这些库中出现的安全漏洞会影响更多的程序，其危害性更大。

4.2 确定目标程序的预期输入

绝大部分可利用的安全漏洞都是由于目标软件未对输入数据进行校验或未对不合法的输入做相应的出错处理。模糊测试是一个不断地枚举输入向量的过程，如果不能充分分析目标程序的输入数据规约，就无法进行有针对性的模糊测试，测试就会有很大的局限性，测试的效果就会大打折扣。比如，对 TCP 协议处理软件进行模糊测试，不仅只应该对数据部分进行测试，序号、确认号、数据偏移字段、标志位、保留字段、窗口、校验和等部分也应该被纳入模糊测试的范围，对不同的软件，在所有的字段中，可以选择性地侧重某些部分，但是一个完整的模糊测试过程应该对所有部分进行充分完全的测试。值得一提的是，以 SPIKE 为首的模糊测试工具能够从大量的输入样本中自主学习目标软件的输入数据规约。

4.3 生成测试用例

在分析目标软件的数据规约的基础上，可以针对性地设计测试用例。测试用例生成是模糊测试的核心和关键步骤。测试用例生成模块主要用于生成测试用例样本，即所谓的模糊器。根据模糊器采用的生成算法可将其分为3类：基于变异的方法、基于生成的方法和两者相结合的方法[13]。基于变异的生成算法从一个合法的样本出发，通过某些算法不断地修改其中一些数据，生成一批畸形的样本；基于生成的算法在对目标软件输入数据格式的规约有很深的了解上，自动生成一些不满足数据规约的测试样本。在生成测试用例过程中，一些启发式的生成策略往往能收到意想不到的效果，如在Windows平台上，可在文件名中包含 &、#、@、\等特殊字符或者使文件名的长度超过MAX_PATH(在Windows中被定义为256)。但是，无论采用哪种方法，测试用例都应该使用自动化的方法来实现。

4.4 执行测试用例

执行测试用例的过程往往与测试用例生成一起执行，即将测试用例输入给目标程序以监测程序的运行状态。试想一下，在对Word程序的模糊测试中存在大批量的样本，手动打开这些样本不仅十分枯燥，还耗费大量不必要的人工时间。通常情况下，模糊测试用例的执行往往采用自动化的方法来实现，由模糊测试系统自动化地执行所有的测试用例。自动化模糊测试的流程如图2所示。

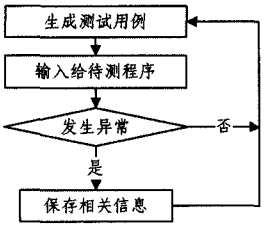

图2 自动化模糊测试流程

每个程序在启动时操作系统均会传递相关的环境变量和命令行，这些信息决定了操作系统如何启动该应用程序。在Windows平台上，一个标准的C++(C)程序的main函数原型如图3所示。

```
#include <iostream>
using namespace std;
int main(int argc, char * argv[], char * * env){
    while( * env){
        cout<< * env<<endl;
        ++env;
    }
}
```

图3 Windows平台main函数原型

当在桌面上双击一个文件时，操作系统实际上会创建一个进程来启动编辑软件并打开相应的文件，创建的进程参数则是编辑软件的路径再加上该文件的路径，各操作系统平台提供的C++(C)编程接口均会提供相应的API函数用于创建进程。在Windows平台上，可以使用Win32 SDK中的CreateProcess函数来自动地执行测试用例，CreateProcess函数的原型如图4所示。

```
BOOL CreateProcess
(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATIONlpProcessInformation
);
```

图4 Win32 CreateProcess函数原型

CreateProcess的第二个参数即是程序的命令行参数，即C++(C)程序中main函数的第二个参数，在遵循POSIX标准的系统平台上也有fork、exec等类似的API。

4.5 异常监视

异常监视是模糊测试过程中一个非常关键的部分，用于记录模糊测试过程中的任何有用的异常信息。试想一下，对程序进行模糊测试的过程中，当执行某个测试用例目标程序发生异常时，倘若没有异常监视模块，测试人员或许根本不知道目标程序发生了异常，更无法找出触发异常的样本及相关信息，整个测试过程将会变得没有任何意义。由于模糊测试过程比较长，当测试用例的数目较多时，不可能由人工监视目标软件是否发现异常，目前异常监视往往采用自动化的方式实现。当前常用的异常监视技术依据原理分为两种：基于调试的方法和基于插装的方法。

基于调试的方法在调试模式下启动目标软件，通过操作系统平台提供的调试API，开发有针对性的异常监测模块，用此方法实现异常监视虽然难度较大，但更加高效。当调试模式在启动目标程序时，启动目标程序的进程称为父进程，被启动的目标程序的执行进程称为子进程，子进程中发生的任何事件在父进程中均可被捕捉，由父进程控制子进程的执行，在父进程中不仅可以获取异常发生时的样本信息，还可以获取异常发生时被调试程序的执行地址、各寄存器值等重要信息[33]。在Windows平台上，创建进程使用Win32 SDK中的CreateProcess API来实现，通过指定CreateProcess API的第6个参数可以使子进程以调试模式启动。

与基于调试的方法相比，基于插装的方法虽然性能有所降低，但其功能却更加强大。在模糊测试过程中，仅仅通过观察程序的输入、输出对了解软件内部的运行信息往往是不够的。如软件运行过程中内部变量的状态信息、模块之间的交互信息等，这些信息对于发现漏洞及定位漏洞特别重要。目前常用的插装方法分为源代码插装、静态代码插装、二进制代码插装等[34]。源代码插装是一种最自然的方式，即在编写软件时，在需要监视的地方插入检测代码，如增加输出信息语句、增加日志语句等，尤其是面向切面编程技术(Aspect oriented Programming)[35]可以较好地用于源代码插装，有效地分离业务逻辑与监测逻辑。静态目标代码插装在Java社区中十分流行，字节码插装可以直接更改中间代码文件(如Java的.class文件等)或在类被类加载器(class loader)装载时进行字节码插装。字节码插装有着执行效率高、插装点灵活

等优点，使其在面向切面编程领域大放光彩，并陆续出现了BCEL[36]、Javassit[37]、ASM[38]等工具。基于二进制的插装技术可以进一步提高模糊测试的异常监测能力，但是其系统消耗较大，且大部分为商业插装软件。常用的二进制插装工具有DynamoRIO[39]、DynInst[40]和Pin[41]等。

4.6 分析异常并确认漏洞

异常分析是模糊测试过程中的最后一步，主要分析目标软件产生异常的位置与引发异常的原因，常用的分析方法借助于IDA PRO、OllyDbg、SoftICE等二进制分析工具进行人工分析。近年来相关学者基于Pin开发离线的二进制代码分析程序，使用二进制代码的动态插装工具对运行中的程序进行监控，将其执行过程中必要的状态信息如线程信息、模块信息、指令信息、异常信息及寄存器上下文等信息[34]，记录到TRACE文件。在TRACE文件的解析过程中，生成函数调用索引、指令地址索引、内存访问索引及异常等索引文件，极大地提高了分析的效率。

5 测试用例生成技术

5.1 测试用例生成技术分类

理论上讲，所有的模糊器都可以归到基于生成、基于变异这两大类的某一类中[13]。在此基础上，众多学者将测试用例生成方法分成以下几类。

5.1.1 预先生成测试用例

首先需要研究目标软件的输入数据规约，理解输入中每个字段支持的数据结构以及可接受的值的范围；然后依据这些已经获得的知识生成用于测试边界条件或是违反归约的测试用例；接着利用这些测试用例来测试该归约实现的完备性。创建这些测试用例可能需要耗费很多精力，但这些用例一旦被创建，就很容易被再次用于测试某种协议或文件格式处理软件的不同实现。PROTOS就是采用这种用例生成方法的代表，PROTOS使用该方法发现了包括CAN-2003-131等众多协议处理软件在处理协议中出现的安全漏洞。值得一提的是，2014年4月互联网爆出与OpenSSL相关的HeartBleed漏洞，该挖掘的工具就是基于PROTOS的。

5.1.2 随机生成测试用例

测试方法是产生一段随机的数据，并将其输入给目标软件，试图使目标软件崩溃或者诱发一些不正常的行为。这种方法可以用来快速地识别目标软件中是否有非常糟糕的代码。随机生成输入在实现上比较简单，却能发现软件中一些严重的错误，同时这种方法比较容易自动化与并行化实现。然而，当前各种软件都会对输入的数据做一些校验，随机生成的数据往往因为校验不通过被目标程序丢弃，在针对WORD程序的模糊测试过程中，相关研究表明，多于85%的测试样本在OLE解析时不通过[13]，无法进行更深入的测试。

5.1.3 变异或强制性生成测试用例

该方法从一个正确有效的协议样本或数据样本开始，不断地打断数据包或是文件中的一个字节、字、双字或是字符串来生成测试用例的测试方法。iDefense公司开发的FileFuzz就是基于这种想法，FileFuzz成功挖掘出了包含MS Office在内的多个软件中存在的安全漏洞。一些学者通过开发复合文档的数据解析模块，从解析一篇合法的复合文档开始，使用系统提供的API修改文档中相应的数据[42-44]，生成测试用例，这种方法避免了传统方法暴力修改导致测试用例不能通过文件校验的缺点，大大提高了模糊测试的效率。同时，文献[28，29]将遗传算法应用到测试用例生成中，从一批合法的数据开始，通过一系列的变异，遗传算法生成新的测试用例，并获取了较理想的实验结果。

5.1.4 自动协议生成测试用例

该方法首先需要对目标软件的输入数据格式进行深入学习与研究，理解和解释协议归约或文件格式定义。在获取知识的基础上，创建一个描述协议规约如何工作的文法。使用这种方法，测试者可以识别出数据包或是文件中的静态部分和动态部分，静态部分是不可随意修改的部分，动态部分是可以被替代的部分。SPIKE[16]和SPIKEfile工具是这类测试工具的代表，两者都采用SPIKE脚本描述目标协议或者文件格式，并使用一个模糊测试引擎来生成测试用例。

5.2 测试用例生成技术研究现状

以FileFuzz为代表的传统模糊测试工具采用“盲测”的方法生成测试用例，即在随机位置插入随机的数据以生成畸形文件。然而现代软件往往使用非常复杂的私有数据结构，如PPT、WORD、EXCEL、MP3、RMVB、PDF、JPEG、ZIP压缩包、带壳的PE文件等。使用的数据结构越复杂，其相应的解析逻辑越复杂，就越容易出现漏洞[27,42]。复杂的数据结构通常具备以下特征：

1）拥有一批预定义的静态数据，如magic、cmd id等；

2）数据结构的内容是可以动态改变的；

3）数据结构之间是嵌套的；

4）数据中存在多种数据关系，如大小校验、指针、引用、CRC校验等。

有意义的数据被编码或压缩，甚至用另一种文件格式来存储，这些格式的文件被挖掘出越来越多的漏洞。对采用复杂数据和结构的复杂文件进行漏洞挖掘，传统的Blind Fuzz暴露出一些不足之处，例如：产生测试用例的策略缺少针对性、生成大量无效测试用例、难以发现文件解析器深层逻辑的漏洞等[8]。

当今测试用例生成技术的革新主要体现在提高数据正确性和提高测试数据的畸形度。

5.2.1 提高测试数据的有效性

早期模糊测试技术的测试数据主要通过随机产生，在广泛应用私有数据和校验和的软件中，这些随机产生的数据往往因为通不过校验而被直接丢弃，根本到达不了存在安全漏洞的高危代码处。为了提高测试数据的合法率及测试数据的代码覆盖率，一些学者对测试数据的生成方法进行研究，产生了多种测试用例的生成思想，大致有如下几种。

1）基于知识发现的测试用例生成技术

基于知识发现的测试用例生成技术是一种通过对目标软件预期的数据输入格式进行分析，了解目标软件所使用的文件格式、网络协议等规约，在此基础上生成满足这些规约关系的测试数据。通过这种办法，可以大大提高生成数据在校验时期的通过率，从而顺利地覆盖可能的高危代码段，发现潜在的安全漏洞。SPIKE、Peach、Sulley[45]等均采用这种测试数据生成方法基于知识发现的测试用例生成技术可分为基于生成的产生方法、基于变异的生成方法[46]和两者结合的方法。

基于生成的产生方法需要对目标软件的预期输入格式非常了解，在深入研究目标软件的文件格式或协议规约的基础上生成违反部分规约的测试数据，这种办法往往需要前期大

量的人工投入[47]。基于变异的产生方法需要对目标软件的预期输入有部分了解，从一个正常的样本出发，修改其中某些字段，从而产生新的畸形样本，这种方法对初始样本的依赖性极大，不同的初始样本往往会带来不同的代码覆盖率以及测试效果[46]。

2)与污点传播相结合的测试用例生成技术

与污点传播相结合的测试用例生成技术的主要思想是应用白盒污点传播技术查找影响不安全函数的输入元素，然后进行有导向的变异数据测试，从而保证其它输入的正确性。这种方法提高了代码的覆盖率，减少了冗余测试用例的生成，提高了漏洞挖掘的效率[47]。BuzzFuzz 是这方面的代表。

3)与静态分析相结合的测试用例生成技术

当前大量程序都使用校验和来判断输入的合法性。针对这一限制，北京大学的王铁磊将符号执行、路径约束求解等分析技术相结合[27,42]，能够自动识别程序中的校验和检验代码，同时使检验失效，这样测试数据才可以绕过校验和检查，当覆盖到不安全函数时，会自动生成带正确校验和的测试数据。

5.2.2 提高测试数据的畸形度

为了使测试数据能够触发程序中的不安全函数，找到软件漏洞，现有两种变异技术可用于提高测试数据的畸形度：1)简单变异技术；2)智能变异技术。

1)简单变异技术

简单变异技术是用预先生成的一些畸形字符串或者随机测试技术产生的畸形数据对目标进行测试。如微软的 P. Godefroid 提出的白盒测试技术[23,24]、意大利的 Andrea Lanz 等提出的白盒测试技术[48]、美国麻省理工的 Vijay Ganesh 等提出的有导向的智能模糊测试技术[47]、中国科学技术大学的 LIU Guang-Hong 等提出的有导向的智能模糊测试技术[49]。

2)智能变异技术

智能变异技术是指能够依据目标不安全函数运行的反馈信息，参照不安全函数的特点，智能地生成更容易触发不安全畸形测试的数据，从而能够以更高的效率、更好的概率生成畸形度高的漏洞触发测试数据。如 FTSG[43]模型应用的有导向的多维变异算子能够根据目标潜在的特点，高效、智能地生成畸形度高的测试数据。

6 模糊测试技术的应用

模糊测试与其它漏洞挖掘方法相比，有着自动化程度高的优点，然而并没有一个通用的模糊测试工具能够对所有类型的程序进行测试，在实际中，往往需要根据不同的测试目标选择不同的模糊测试工具和模糊测试方法。下面，根据不同的测试对象对模糊测试技术进行分类。

6.1 文字处理软件的模糊测试

几乎所有的电脑、智能手机等设备上都包含文字处理程序，例如办公软件通常需要处理 DOC、XLS、PPT 等类型的文件，当办公软件打开一个精心构造的恶意文件时，有可能触发其中存在的安全漏洞。自 2004 年从 Windows 平台上 JPEG 处理引擎中一个缓冲区溢出漏洞开始，文件处理软件的安全便受到广大安全工作者的关注，针对文件处理软件的模糊测试相关技术研究也逐渐拉开序幕。随后，用于文件处理软件的模糊测试工具诞生，FileFuzz、SPIKEfile、notSPIKEfile、PAIMEIfilefuzz[50]是其中的佼佼者。

6.2 网络协议的模糊测试

互联网的兴起使得网络协议的使用越来越广泛，常见的协议如 HTTP、FTP、SSL 等已成为互联网不可分割的一部分。互联网的公开性与联通性决定了网络协议处理软件漏洞的巨大危害性，2014 年互联网上爆出与 OpenSSL 相关的 HeartBleed 漏洞，利用该漏洞，攻击者可以获取购物者的帐号、密码等秘密信息，一时间各大购物网站的交易量急剧下降，HeartBleed 漏洞严重威胁购物者的帐号安全，同时也给各大电商集团带来了巨大损失。网络协议根据其复杂度可分为两个主要类别：简单协议和复杂协议。简单协议通常使用简单的认证或没有认证，简单协议通常是面向字符的，不包含校验或长度信息，FTP、HTTP 是这类协议的代表。复杂协议通常使用二进制数据，认证过程比较复杂，微软的 RPC 以及广泛使用的 SSL 协议是其中的代表。

常用的网络协议模糊测试器主要有 SPIKE 和 Peach。SPIKE 是第一个公开的网络协议模糊测试工具，该工具包含一些预生成的针对几种常用协议测试的测试用例集，同时，SPIKE 以开放 API 的形式提供使用接口，方便用户定制。Peach 是一个使用 Python 编写的跨平台模糊测试框架，该工具比较灵活，使用一个 XML 文件引导整个测试过程，几乎可以用来对任何网络协议进行模糊测试。

6.3 Web 应用程序的模糊测试

随着 Web 2.0 的兴起，传统的一些桌面应用已被一些 Web 应用所取代。Web 应用是采用 B/S 架构，按照 HTTP/HTTPS 协议提供服务的统称。由于国内外对 Web 安全的研究起步较晚，目前 Web 应用中存在着大量拒绝服务(Denial-of-service，DoS)[51]、跨站脚本(Corss-site scripting，XSS)[52]、SQL 注入(SQL injection)[53]等类型的安全漏洞。Web 应用的模糊测试不仅能发现 Web 应用本身的漏洞，还可以发现 Web 服务器和数据服务器中的漏洞。目前主要用于 Web 应用的模糊测试工具有 SPIKE Proxy、WebScarab[54]、Web Inspect 等。SPIKE Proxy 是 Dave Aitel 使用 Python 基于 GPL 许可开发的基于浏览器的 Web 应用模糊测试工具，它以代理的方式工作，捕获 Web 浏览器发出的请求，允许用户针对目标 Web 站点运行一系列的审计工作，最终找出多种类型的漏洞。WebScarab 是 Web 应用安全项目(OWASP)开发的多种用于测试 Web 应用安全的工具中出色的一个。Web Inspect 是 SPI Dynamics 公司开发的一款商业工具，提供了一整套全面测试 Web 应用的工具[53]。

6.4 Web 浏览器的模糊测试

严格意义上讲，Web 浏览器是文件处理软件中的一种，主要处理的文件是诸如 HTML、CSS、JS 之类的与 Web 相关的文件，因其特殊性，暂时将其分成单独一类。对 Web 浏览器的模糊测试不应局限于 HTML、CSS、JS 等类型文件的解析模块上，类似 ActiveX 这样的插件也应该是模糊测试的一个重要组成部分。

与 Web 相关的模糊测试工具有 COMRaider[19]、mangleme、Hamachi、CSSDIE 等。COMRaider 主要用于 IE 浏览器中广泛使用的 ActiveX 控件的模糊测试；mangleme 是第一个发布的针对 HTML 的模糊测试器；与 mangleme 类似，Hamachi 主要的目标是动态 HTML(DHTML)；CSSDIE 则主要用于测试 Web 页面布局中的 CSS 文件。

特别需要指出的是：所有的应用程序都在某个具体的环

境中运行，每个程序在作为一个进程被操作系统启动时，都有自己独立的命令行参数和环境变量，大量的程序本身往往假设命令行参数和环境变量是合法的而直接使用。因此，对命令行参数和环境变量进行模糊测试很有必要，以下对这两者进行简单的介绍。

1)命令行的模糊测试

一个应用程序开始运行时，通常会处理用户传给它的命令行参数，命令行参数一般由操作传递给应用程序，由应用程序决定如何使用。命令行参数的使用一般比较简单，以至于大多程序不对其作任何检查就直接使用，以图 5 所示的程序为例。

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[]){
for(int i=1;i<argc;++i){
system(argv[i]);
}
}
```

图 5　命令行可能存在的安全漏洞示例代码

在这个程序的 system 函数调用中，并不对命令行参数进行任何检查，如果命令行中包含一个木马程序的路径，则这个程序就会自动执行木马程序。由此可见，不对命令行参数进行任何检查可能导致严重灾难。

clfuzz 和 iFUZZ 是命令行模糊测试器中的代表。clfuzz 主要用于测试应用中的格式字符串和缓冲区溢出安全漏洞；iFUZZ 的使用更加灵活，其测试目标与 clfuzz 大致相同。

2)程序环境变量的模糊测试

程序运行时，操作系统会为其创建一个进程，同时，在进程中会有一些操作系统安排好的变量，当前程序目录、环境变量等被包含在其中。环境变量通常是一个 key-value 二元组序列。在 C 语言中通常使用 getenv[38] 函数获取 key 变量对应的 value，在 Windows 中，可以使用 Win32 API GetEnvironmentStrings[38] 或者 C 语言 main 函数的第三个参数获取函数变量。与命令行参数类似，环境变量的使用一般比较简单，大多数程序也不做任何处理就使用，以图 6 所示程序为例。

```
#include <iostream>
#include <stdlib.h>
using namespace std;
void func(){
    char buf[100];
    memset(buf,0,sizeof(buf));
    strcpy(bug,getenv("AppPath"));
    printf("%s\n",buf);
}
int main(){
    func();
}
```

图 6　环境变量可能造成缓冲区溢出的代码示例

此程序的主要目的是将 AppPath 对应的 value 复制到 buf 缓冲区中，然后输入。然而，在代码里面存在明显的缓冲区溢出漏洞，并没有对 AppPath 对应的字符串长度进行判断，如果 AppPath 对应的字符串的长度超过 100，则会造成缓冲区溢出，可能导致灾难性的后果。

Sharefuzz[55] 和 iFuzz 是环境变量模糊测试器中杰出的代表。Sharefuzz 是第一个公开的环境变量模糊测试器；iFUZZ 虽然属于命令行模糊测试器，但同时也对环境变量进行了大量的测试，同时自定制性更强。

7　模糊测试的发展方向

模糊测试有其自身的优势，与此同时其不足也是十分明显的。模糊测试技术仍然存在许多局限性，目前使用简单的模糊测试已经很难发现新类型的高危漏洞。通过模糊测试技术挖掘的漏洞仍大多是传统的堆溢出、栈溢出等溢出类漏洞，对于后门、验证绕过、多阶段触发等类型的漏洞却无能为力。更重要的是，模糊测试并不能保证覆盖到所有的语句分枝，目前模糊测试的自动化程度还不高。如何克服缺陷、提高模糊测试的效率是模糊测试未来研究的热点。模糊测试未来研究的热点可能包括以下几个方面。

7.1　提高模糊测试的自动化程度

模糊测试看似简单，但它能揭示出程序中的重要漏洞，通过模糊测试挖掘到的漏洞通常都比较严重。模糊测试的主要问题在于它是一种盲注入的方式，手工进行模糊测试工作量过大，甚至几乎无法进行比较全面的测试，导致其实用价值受到很大影响，在常用的模糊测试框架中，目标软件的漏洞成因、目标软件的输入数据的规约、生成测试用例等往往需要人工参与。目前大量的关注点在于研究自动化与智能化的模糊测试方法。

7.2　优化测试用例生成策略

模糊测试中关键的步骤是测试用例生成，传统的模糊测试在生成测试用例时往往盲目地去变异正常测试用例中某一部分，这种盲目的变异方法造成测试用例规模可能达到十万、百万的级别，但其测试效果并不理想。因此，测试用例生成策略的设计与改进是目前模糊测试技术的热点研究内容之一。引入退火遗传算法等对测试规则进行改进，使得最小测试用例集合能够覆盖最大的代码执行路径，以发掘那些隐藏较深的软件漏洞，是目前模糊测试技术的发展趋势[47,48]。同时，传统的测试用例生成策略对多点触发型漏洞无能为力，近年来，多维模糊测试技术的兴起使得用多点触发漏洞的挖掘成为可能。多维模糊测试技术是指对多维输入同时变异的模糊测试技术，其能够挖掘到传统模糊测试技术无法挖掘到的漏洞。然而，多维模糊测试存在脆弱函数查找、脆弱函数覆盖等问题，如何克服这些困难，将是未来多维模糊测试技术中的一个热点研究问题。

7.3　与虚拟机技术相结合

与虚拟机技术相结合，能够快速、全面地获取不安全函数、目标程序的更多反馈信息，进而生成畸形度更高、正确度更高的测试数据，改善漏洞挖掘的效果。

7.4　模糊测试效果的评估技术研究

以 SAGE 为代表的模糊测试框架以代码覆盖率来评价模糊测试效果，测试的过程缺乏聚焦性，测试路径通常会随意扩散，这往往是不直接、不科学的；从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测试效果会更加科学，也能更好地指导测试用例的生成和模糊测试技术的进一步发展。

结束语 随着各行业信息化的发展，软件无处不在，其重要性日益显著，因而软件的安全性显得尤为重要。作为一种漏洞挖掘技术，模糊测试具有其它漏洞挖掘技术所不具备的优点，是软件安全领域研究的重要方向，国内外众多研究者在模糊测试领域进行了大量有益的探索和研究工作。

本文对模糊测试技术作了全面的回顾与总结。首先将其它漏洞挖掘技术与模糊测试技术进行了对比；然后详细介绍了模糊测试研究进展与模糊测试过程，着重阐述了模糊测试中的测试用例生成技术，并介绍了模糊测试在各类软件安全性测试中的应用情况；最后对模糊测试技术未来可能的研究方向进行了展望。

总之，我们认为在未来相当长的一段时间内，模糊测试仍将是软件漏洞挖掘的一种实用而有效的利器。

参 考 文 献

[1] CNCERT. 2013 China Internet Network Security Report [M]. Beijing: Post & Telecom Press, 2013(in Chinese)
国家计算机应急技术处理协调中心. 2013 年中国互联网网络安全报告[M]. 北京：人民邮电出版社，2013

[2] Mei Hong, Wang Qian-xiang, Zhang Lu, et al. Analysis of the progress of software technology[J]. Chinese Journal of Computers, 2009, 32(9): 1697-1710(in Chinese)
梅宏，王千祥，张路，等. 软件分析技术进展[J]. 计算机学报，2009, 32(9): 1697-1710

[3] ITS4[EB/OL]. http://seclab.cs.ucdavis.edu/projects/testing/tools/its4.html

[4] FLAWFINDER[EB/OL]. http://www.dwheeler.com/flawfinder

[5] IDA PRO[EB/OL]. https://www.hex-rays.com/index.shtml

[6] Zhao Xiao-dong. Research and implementation of based malware analysis tool[D]. Nanjing: Nanjing University, 2012(in Chinese)
赵晓东. 基于虚拟化的恶意软件分析工具的研究与实现[D]. 南京：南京大学，2012

[7] Vouk M A. Software reliability engineering [OL]// A Tutorial Presented at the Annual Reliability and Maintainability Symposium. 2000. http://renoir.csc.ncsu.edu/Faculty/Vouk/vouk_se.html

[8] OllyDbg[EB/CP]. http://www.ollydbg.de

[9] WinDbg[EB/CP]. Whttp://www.windbg.org

[10] SoftICE[EB/OL]. http://en.wikipedia.org/wiki/SoftICE

[11] Miller B P, Koski D, Lee C, et al. Fuzz Revisited: A Reexamination of the Reliability of UNIX Utilities and Services[R]. Wisconsin: Computer Sciences Department, University of Wisconsin, 1995

[12] Miller B P, et al. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33: 32-44

[13] Cohen M B, Snyder J, Rothermel G. Testing across configurations: implications for combinatorial testing[J]. ACM SIGSOFT Software Engineering Notes, 2006, 31(6): 1-9

[14] Shu G, Lee D. Testing security properties of protocol implementations-a machine learning based approach[C]// 27th International Conference on Distributed Computing Systems (ICDCS'07). IEEE, 2007: 25

[15] Aitel D. The advantages of block-based protocol analysis of security testing[R]. New York: Immunity Inc, February 2002

[16] SPIKE[EB/OL]. http://www.immunitysec.com/resources-freesoftware.shtml

[17] Peach[EB/OL]. http://peachfuzz.sourceforge.ne

[18] FillFuzz[EB/OL]. http://labs.idefense.com/software/fuzzing.php

[19] COMRaider[EB/OL]. http://labs.idefense.com/software/fuzzing.php#more_comraider

[20] AxMan[EB/OL]. http://metasploit.com/users/hdm/tools/axman

[21] Demott J. The evolving art of fuzzing[S]. Defcon, 2006

[22] DeMott J, et al. Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing[S]. BlackHat and Defcon, 2007

[23] Godefroid P, et al. Grammar-based whitebox fuzzing[C]// 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation 2008 (PLDI'08). Tucson, AZ, United states, 2008: 206-215

[24] Godefroid P, et al. Automated Whitebox fuzzing[C]// Proc Network Distributed Security Symposium(NDSS). San Diego, California, 2008

[25] Ganesh V, et al. Taint-based directed whitebox fuzzing [C]// Proc 2009 31st International Conference on Software Engineering(ICSE 2009). Vancouver, BC, Canada, 2009: 474-484

[26] Wang T L. Research on key technologies of vulnerability exploiting for binary programs[M]. Beijing: Peking University, 2011 (in Chinese)
王铁磊. 面向二进制程序的漏洞挖掘关键技术研究[M]. 北京：北京大学，2011

[27] Wang T, et al. TaintScope: A checksum-aware directed fuzzing tool for automatic softwarevulnerability detection [C]// Proc 31st IEEE Symposium on Security and Privacy, SP 2010. Berkeley/Oakland, CA, United states, 2010: 497-512

[28] Zhang Shu-qin. Research on Fuzz testing technology based on genetic algorithm[D]. Wuhan: Huazhong University of Science and Technology, 2011(in Chinese)
章淑琴. 基于遗传算法的模糊测试技术研究[D]. 武汉：华中科技大学，2011

[29] Du Xiao-jun, Lin Bo-gang, Lin Zhi-yuan, et al. Research of Multi population genetic algorithm in fuzz testing [J]. Journal of Shandong University(Science Edition), 2013(7): 79-84(in Chinese)
杜晓军，林柏钢，林志远，等. 安全软件模糊测试中多种群遗传算法的研究[J]. 山东大学学报(理学版)，2013(7): 79-84

[30] SecurityFocus[EB/OL]. http://www.securityfocus.com/

[31] Secunia [EB/OL]. http://secunia.com/

[32] CNVD[EB/OL]. http://www.cnvd.org.cn/

[33] Jalbert N, Sen K. A trace simplification technique for effective debugging of concurrent programs [C] // Proceedings of the eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2010: 57-66

[34] Luk C, Cohn R, Muth R, et al. Pin: Building customized program analysis tools with dynamic instrumentation[J]. Programming Language Design & Implementation, 2005, 40(6): 190-200

[35] Kojarski S, Lorenz D H. Pluggable AOP: designing aspect mechanisms for third-party composition[J]. Oopsla'05 Proceedings of Annual Acm Sigplan Conference on Object Oriented Programming Systems Languages & Applications, 2005, 40(10): 247-263

[36] BCEL[EB/OL]. http://commons.apache.org/proper/commons-bcel/

(下转第 26 页)

从图 14 可看出，在没有进一步优化前，线程交换获得的加速效果并不理想，对于某些输入集，线程交换引起的开销过大，导致加速比小于 1。进一步优化后，开销得到大幅度减小，加速比提高，在交换范围为 256 和 512 时，性能分别最高提高了 6.4%和 5.3%。从图 15 可看出，通过线程交换，程序的功耗得到一定程度的降低，而进一步优化后，功耗变化不大。

结束语 本文首先研究了线程交换对程序性能和功耗的影响。通过实验发现，线程交换在减少 Branch Divergence 的同时会改变程序的访存行为，产生访存开销，这会在一定程度上影响由于 Branch Divergence 减少带来的收益，甚至完全抵消其收益。功耗方面，对于一些程序来说，线程交换影响访存行为并不一定会引起访存次数的增多，因此线程交换并不一定会使访存部分的功耗增加，并且由于线程交换减少了某些功能单元的访问次数，一般来说优化后程序的功耗会降低。

其次，本文研究了交换范围对程序性能和功耗的影响。不同的交换范围意味着不同的优化程度，一般来说，交换范围越大，优化程度越高，同时带来的开销也越大，所以并不是交换范围越大越好，如何选择合适的交换范围往往需要具体问题具体分析。

最后，本文选择 Reduction 以及 Bitonic 这两个含有较多 Branch Divergence 的程序进行优化和性能测试，研究了它们在不同交换范围下性能和功耗的表现，并针对优化过程中存在的问题进行了进一步分析及优化。通过优化，Reduction 程序在性能平均损失 4%的情况下功耗最大降低了 7%，Bitionic 程序在交换范围为 256 和 512 时性能分别最高提升了 6.4%和 5.3%。

接下来，将通过更多程序测试本文所提优化方法的效果，并将该优化策略集成在面向 CUDA 的编译器中。

参 考 文 献

[1] NVIDIA CUDA[EB/OL]. [2015-5-15]. http://www.nvidia.com/cuda

[2] Zhang E Z, Jiang Yun-lian, Guo Zi-yu, et al. On-the-Fly Elimination of Dynamic Irregularities for GPU Computing[C]// Proceedings of the 16th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS). Newport Beach, CA, USA, ACM, 2011: 369-380

[3] Zhang E Z, Jiang Yun-lian, Guo Zi-yu, et al. Streamlining GPU application on the fly: thread divergence elimination through runtime thread-data remapping [C]// Proceedings of the 24th International Conference on Supercomputing. Tsukuba, Ibaraki, Japan, ACM, 2010: 115-126

[4] Han T Y D, Abdelrahman T S. Reducing Branch Divergence in GPU Programs[C]// Proceedings of the 4th Workshop on General Purpose Processing on Graphics Processing Units. Newport Beach, CA, ACM, 2011: 1-8

[5] Qian Cheng, Shen Li, Zhao Xia, et al. Thread Swapping Based Optimization Strategy for Sort Algorithms on GPUs[J]. Journal of Northeastern University (Natural Science), 2014, 35(1): 68-73(in Chinese)
钱程，沈立，赵夏，等. GPU 上基于线程交换的排序算法优化策略[J]. 东北大学学报(自然科学版)，2014，35(1)：68-73

[6] Bakhoda A, Yuan G L, Fung W W L, et al. Analyzing CUDA workloads using a detailed GPU simulator[C]// IEEE International Symposium on Performance Analysis of Systems and Software. Boston, MA, USA, IEEE, 2009: 163-174

[7] GPGPU-Sim Manual[EB/OL]. [2015-5-15]. http://gpgpusim.org/manual/index.php/GPGPU-Sim_3.x_Manual

[8] Xu Qiu-min, Annavaram M. PATS: Pattern Aware Scheduling and Power Gating for GPGPUs[C]// Proceedings of the 23rd international conference on Parallel Architectures and Compilation. Edmonton, AB, Canada, ACM, 2014: 225-236

[9] Leng Jing-wen, Tayler H, Ahmed E, et al. GPUWattch: Enabling Energy Optimization in GPGPUs[C]// Proceedings of 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel, ACM, 2013: 487-498

[10] GPUWattch[EB/OL]. [2015-5-15]. http://gpgpu-sim.org/gpuwattch

(上接第 8 页)

[37] Javassit[EB/OL]. http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/html/javassist/CtClass.html

[38] ASM[EB/CP]. http://asm.ow2.org/

[39] DynamoRIO[EB/OL]. http://www.dynamorio.org/

[40] DynInst[EB/OL]. http://www.dyninst.org/

[41] Pin [EB/OL]. https://software.intel.com/sites/landingpage/pintool/docs/71313/Pin/html/

[42] Wang T, Wei T, Zou W. Checksum-Aware Fuzzing Combined with Dynamic Taint Analysis and Symbolic Execution[J]. Acm Transactions on Information & System Security, 2011, 14(2): 613-613

[43] Atwood J W, et al. A new fuzzing technique for software vulnerability mining[D]. Concordia University, 2009

[44] MSDN[EB/OL]. https://msdn.microsoft.com/zh-cn/default.aspx

[45] Sulley[EB/OL]. http://resources.infosecinstitute.com/sulley-fuzzing/

[46] Miller C, Peterson Z N J. Analysis of Mutation and Generation-Based Fuzzing[R/OL]. http://securityevaluators.com/files/papers/analysisfuzzing.pdf

[47] Ganesh V, Leek T, Rinard M. Taint-based Directed Whitebox Fuzzing[C]// Proceeding International Conference on Software Engineering. 2009: 474-484

[48] Lanzi A, Martignoni L, Monga M, et al. A Smart Fuzzer for x86 Executables[C]// ICSE Workshops 2007, Third International Workshop on Software Engineering for Secure Systems, 2007 (SESS'07). IEEE, 2007: 7-7

[49] Liu G H, Wu G, Tao Z, et al. Vulnerability Analysis for X86 Executables Using Genetic Algorithm and Fuzzing[C]// International Conference on Convergence & Hybrid Information Technology. IEEE, 2008: 491-497

[50] PaiMei[CP/OL]. http://www.openrce.org/downloads/details/208/PaiMei

[51] Liu W. Research on DoS Attack and Detection Programming[C]// Workshop on Intelligent Information Technology Applications IEEE. 2009: 207-210

[52] Hydara I, Sultan A B M, Zulzalil H, et al. Current state of research on cross-site scripting (XSS)-A systematic literature review[J]. Information & Software Technology, 2015: 170-186

[53] Dukes L, Yuan X, Akowuah F. A case study on web application security testing with tools and manual testing[C]// Southeastcon, IEEE. IEEE, 2013: 1-6

[54] WebScara[EB/OL]. https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

[55] ShareFuzz[CP/OL]. http://www.immunitysec.com/resources