doi :10.3969/j.issn.1671-1122.2019.01.001

一种基于随机探测算法和信息聚合的漏洞检测方法

文伟平[1]，李经纬[1]，焦英楠[2]，李海林[1]

（1. 北京大学软件与微电子学院，北京 102600；2. 国家计算机网络应急技术处理协调中心，北京 100029）

摘　要：随着计算机软件复杂度的持续增长，软件架构的安全性不断下降。由于软件各模块耦合性过高，导致软件漏洞数量急剧增加，安全漏洞的检测和防护技术逐渐成为网络安全领域的重点研究方向。现有的漏洞静态检测方法检测效果较差，而模糊测试技术需要消耗大量时间，业内缺乏能够快速对大规模二进制程序进行漏洞扫描的方法。文章基于机器学习方法，使用一种随机探测算法对反编译后的程序进行轻量级静态特征提取，并在动态特征提取过程中对参数进行信息聚合，对提取到的动态特征和静态特征分别运用 Text-CNN、Logistic、随机森林等算法进行模型训练。实验表明，文章方法可以有效对二进制程序进行漏洞检测。

关键词：漏洞检测；特征提取；机器学习

中图分类号：TP309　**文献标识码：**A　**文章编号：**1671-1122（2019）01-0001-07

中文引用格式：文伟平，李经纬，焦英楠，等 . 一种基于随机探测算法和信息聚合的漏洞检测方法 [J]. 信息网络安全，2019，19（1）：1-7.

英文引用格式：WEN Weiping，LI Jingwei，JIAO Yingnan，et al. A Vulnerability Detection Method Based on Random Detection Algorithm and Information Aggregation[J].Netinfo Security, 2019,19（1）：1-7.

A Vulnerability Detection Method Based on Random Detection Algorithm and Information Aggregation

WEN Weiping[1], LI Jingwei[1], JIAO Yingnan[2], LI Hailin[1]

(1. *School of Software and Microelectronics*, *Peking University*, *Beijing* 102600, *China*; 2. *National Computer Network Emergency Response Technical Team / Coordination Center*, *Beijing* 100029, *China*)

Abstract: As the complexity of computer software continues to grow, the security of software architectures continues to decline. Due to the high coupling of software modules, the number of software vulnerabilities has increased dramatically. The detection and protection technologies of security vulnerabilities have gradually become key research directions in the field of network security. However, the existing vulnerability detection methods have many shortcomings. Fuzzy testing technology consumes a lot of time, and

收稿日期：2018-10-16

基金项目：国家自然科学基金 [U1736218]

作者简介：文伟平 (1976—)，男，湖南，教授，博士，主要研究方向为网络攻击与防范、恶意代码研究、信息系统逆向工程和可信计算技术等；李经纬 (1995—)，男，辽宁，硕士研究生，主要研究方向为漏洞分析和漏洞挖掘；焦英楠 (1983—)，女，辽宁，工程师，硕士，主要研究方向为软件工程、信息安全等；李海林 (1993—)，男，四川，硕士研究生，主要研究方向为软件工程。

通信作者：文伟平 weipingwen@ss.pku.edu.cn

there is no fast vulnerability scanning method for large-scale binary programs in the industry. Based on machine learning method, this paper uses a random detection algorithm to extract lightweight static features of decompiled programs, and aggregates parameters in the process of extracting dynamic features. Text-CNN, Logistic and random forest algorithms are used to train dynamic and static features respectively. Experiments show that this method can effectively detect vulnerabilities in binary programs.

Key words: vulnerability detection; feature extraction; machine learning

0 引言

随着计算机技术的不断发展，计算机软件复杂度也在不断增长，软件内部的安全漏洞数量急剧增加。在计算机软件应用遍布社会各行业的同时，软件具有的安全问题也给社会带来巨大经济损失，其中计算机操作系统漏洞造成的危害尤为严重。

安全漏洞 [1] 是指程序代码实现中存在的具体错误，如逻辑错误、边界检查错误等。漏洞本身并不会造成攻击，执行过程中可能也并不会出现错误，因此在某些情况下，漏洞的存在对程序的运行并没有什么影响。但是一旦漏洞被攻击者知晓，攻击者便可根据这个漏洞编写相应的 shell code，并利用社会工程学方法将此 shell code 散播出去，产生违法犯罪行为，如越权访问机密数据、窃取用户隐私数据、破坏计算机信息系统等。因此安全研究人员应尽可能地发现操作系统及应用程序中的一些未知漏洞，并对相应漏洞打补丁，以提高操作系统和应用程序的安全性。综上所述，对操作系统及应用软件进行全面的漏洞挖掘是必不可少的。现在任何一款操作系统基本都包含成千上万个二进制程序，然而缺乏开源的能够快速对大规模二进制程序进行漏洞扫描的检测方法。本文基于机器学习方法，利用轻量级静态特征与动态特征，设计并实现了一种针对内存损坏类漏洞 [2] 的漏洞检测方法。

1 相关工作

漏洞检测指漏洞安全研究人员通过技术手段在漏洞被利用之前尽可能地发现程序中的漏洞。漏洞检测技术的发展经历了手工测试、程序源码分析、程序二进制代码分析，以及污点分析、符号执行、模糊测试等过程 [3]，其自动化过程也经历了人工检测、编译器半自动化检测、全自动化检测等过程。如今的全自动化技术已被成熟应用于工业界的漏洞检测，但对于新型漏洞的挖掘，手工测试依然是最有效的方法。

基于静态分析的漏洞检测工具通常被用来检测软件内部的缺陷，常用的静态检测工具主要有 ITS4[4]、cpplint[5]、pylint[6] 等。尽管这些检测工具为漏洞检测提供了方便，然而漏洞检测的效果较差。因此有研究者进行轻量级的静态分析研究 [7-9]，但无论是源码分析还是二进制代码分析，这种技术通常都只关注某种特定的漏洞。此外，这些轻量级的静态分析技术还以牺牲鲁棒性来减低误报率，提升检测效果。一些静态分析技术已经被证明可以成功找到经典的程序缺陷，如 buffer 溢出或空指针解引用，但误报率很高，并且只有少数检测工具能够对二进制代码进行检测 [10]。

因此，最有效的漏洞检测技术之一仍然依赖于大规模的模糊测试 [11]，为目标程序提供各种输入并监视异常结果以发现软件漏洞。黑盒模糊技术容易实施，且具有高度的可扩展性，不涉及复杂的计算和繁重的程序监控技术。但是这种方法不允许控制程序的执行，且为获得有价值的测试结果需要进行大量模糊测试，此外程序运行异常的后续处理比较复杂。为了克服这些限制，有研究者提出了白盒模糊测试方法。其基本思想是在代码的帮助下生成应用程序的输入。在白盒模糊测试方向上已经开发了许多工具，如 Klee[12]、TaintScope[13]，并在一些案例研究中展现了良好的漏洞挖掘能力。此外，也有研究者利用 concolic 执行漏洞检测 [14]。

模糊测试技术非常耗时，在有限时间内对操作系

统进行大规模的模糊测试基本是不可能的。为了提高检测速度，利用机器学习技术进行漏洞挖掘被提上了日程。文献 [15] 通过分析并提取恶意软件中的静态和动态特征形成二进制特征向量，并以该特征向量进行漏洞检测。但其静态和动态特征的提取属于重量级，内存开销较大。文献 [16] 通过分析易受攻击的源代码来寻找易受攻击的代码模式，进而提取相关特征，并利用机器学习方法进行漏洞检测，但特征提取过程需要程序源码的参与。近年来也出现了其他很多利用机器学习方法进行攻击检测的技术 [17,18]，如入侵检测系统。利用机器学习方法进行漏洞检测已逐渐成为业内的研究热点。

2 漏洞检测方法原型设计

本文从 Debian 漏洞跟踪报告中获得 1400 个二进制程序。首先对这些二进制程序进行轻量级的静态特征与动态特征提取，得到基于函数调用子序列的静态特征集和基于程序执行轨迹的动态特征集；然后利用 logistic 回归、基于 CNN 的文本模型（Text-CNN）与随机森林 3 种机器算法对静态特征集和动态特征集进行正负样本分类的模型训练。在此基础上设计并实现了一种基于机器学习的漏洞检测方法，用以对操作系统中的大规模二进制程序进行漏洞检测。漏洞检测方法流程如图 1 所示。

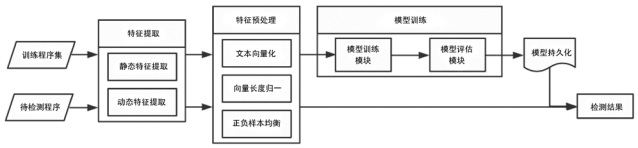

图 1 漏洞检测方法流程

2.1 基于随机探测算法的静态特征提取

静态特征的提取主要是为了在不运行程序的情况下提取出漏洞相关的关键信息。通常静态特征是基于图结构的，如数据流图、程序调用图、控制流图，然而这样的图结构不是很方便用于模型训练。可以利用轻量级静态分析技术直接从二进制程序中提取静态特征，如图 2 所示。首先利用线性扫描算法将二进制程序进行反汇编；然后将反汇编后的汇编代码与 C 标准库函数进行比对，从汇编代码中提取出 C 函数集；接着从该 C 函数集中随机选取一个函数地址建立函数调用序列，并进行一次随机探测，得到程序控制流图，作为一个静态特征。随机探测算法流程如图 3 所示。

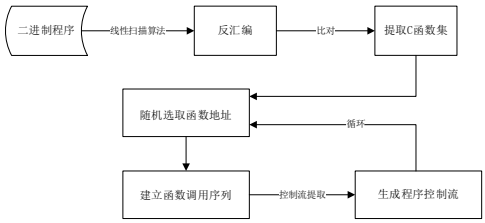

图 2 静态特征提取流程

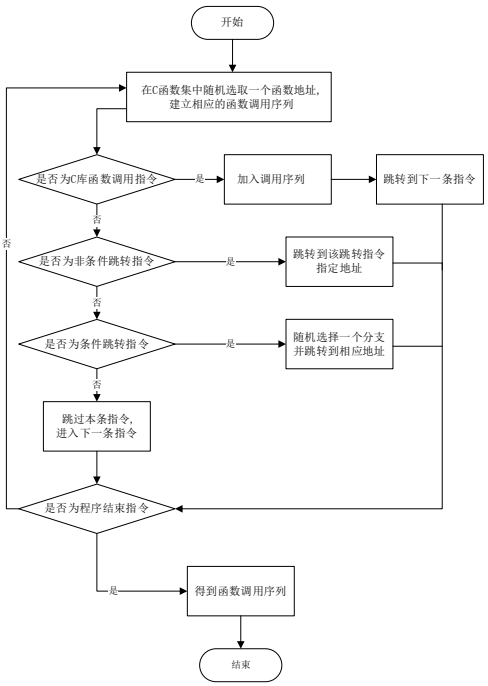

图 3 基于随机探测算法的静态特征提取流程

由图 3 可知，静态特征随机探测从程序反汇编得到的 C 函数集中某一个函数地址开始。如果该函数地址所对应的汇编指令为 C 库函数调用指令，则将该库函数记录到静态特征序列中，并跳转到下一条指令；若该指令不是 C 库函数调用指令，则继续判断该指令是否为非条件跳转指令。如果是，则跳转到跳转指令指定的地址；否则判断该指令是否为条件跳转指令。如果是，则随机选择一个分支，并跳转到相应地址；否则表明该指令既不是条件或非条件跳转指令，也不

是 C 库函数调用指令，那么直接进入下一条指令。最后判断接下来的指令是否为程序结束指令，如果是，则结束特征提取，取出保存的函数调用序列；否则按上述判断过程继续在 C 函数集中随机选取下一个函数地址进行特征提取，直到提取完 C 函数集中所有函数的调用序列特征。

本文通过轻量级的静态特征提取方法抽取了多条 C 库函数调用序列，以达到模拟图结构的效果。这样抽取出的静态特征不仅方便用于模型训练，且特征辨识度较高。最终提取出了 75032 条不同的函数调用序列作为机器学习模型训练所用的静态特征。

2.2 基于信息聚合的动态特征提取

动态特征提取主要是 hook 程序的关键代码，在有限时间内执行程序，提取出程序执行事件序列作为机器学习模型训练所用的动态特征。因此动态特征的提取也就是对一系列程序执行事件的收集和顺序存储。程序执行事件是指程序对 C 标准库函数的一次实参调用或指程序结束。可以利用带实参的函数或程序返回状态来表示程序执行事件，如公式（1）所示。

$$f_{C_i}(\text{agr1},\text{arg2},...,\text{arg}n)|\text{FinalState} \quad (1)$$

其中，f_{C_i} 表示标号为 i 的 C 标准库函数，agr_i 表示该函数的第 i 个参数，FinalState 表示程序的结束状态。程序结束状态包含该程序最后一次对 C 库函数调用的相关信息，往往与内存泄露漏洞密切相关。程序结束状态分为程序退出（Exit）、异常终止（Crash）、诱发性异常终止（Abort）和时间耗尽（Timeout）4 种。

与静态特征提取不同的是，动态特征提取可能从测试用例中提取出大量事件。即使对于小型程序来说，所提取的事件序列也可以构造出非常大的数据集，如简单的循环就可能被展开为任意长的事件序列。机器学习分类器如果将这种未经处理的事件序列用于模型训练，会给模型学习带来很大困难，主要有两点原因：1）参数的值域比较大，因此存在大量不同的参数值（例如，32 位参数可能取值有 232 种）。如果想用不连续的事件序列来训练机器学习模型，就必须大幅度减小参数的取值范围。2）参数的特征值较多也会使单个参数携带信息较少，导致所有参数携带的信息较为分散。

为解决上述两个问题，可以使用合适的子类型来标记每个参数，如图 4 所示。

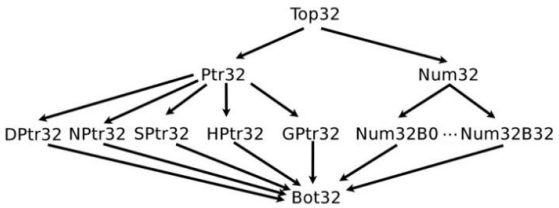

图 4 动态特征参数子类型

图 4 中，对指针 Ptr32 进行了划分。其中，HPtr32 为堆地址，SPtr32 为栈地址，GPtr32 为全局内存地址，DPtr32 为悬空指针，NPtr32 为空指针。整数 Num32 传递的信息比指针少，根据整数的取值范围可以对其进行子类型的划分。例如，C 标准库函数 fread() 的参数取值过大会导致内存损坏，根据取值范围对 fread() 的参数进行子类型划分后，可保留该参数取值与内存损坏漏洞间的关系，形成有效应用于漏洞检测的动态特征。图 4 中，Num32Bn 指示在 2n 和 2n+1 之间的一个 32 位数。

采用子类型标记方法对参数信息进行聚合有效克服了参数信息的分散问题。利用该方法，本文提取了 142325 条不同的程序执行轨迹。

2.3 特征预处理与机器学习模型训练

对样本的特征集采用 tf-idf 算法进行特征向量化，将特征向量输入模型。由于采用 tf-idf 算法进行特征向量化后，向量长度已经一致，因此不需要再做长度归一处理。

分别采用 logistic 回归模型、随机森林与 Text-CNN 对前文得到的静态特征集和动态特征集进行模型训练。logistic 回归模型是一个线性分类器，只能解决平面可分的数据集，为了防止过拟合，为其代价函数加入了 L2 正则。随机森林基于 100 棵 GDBT 树的 bagging 方法；Text-CNN 将窗口设置为 4×1，kernel 数设置为 50、100、200，通过最大化训练数据上的对数似然函

数来求解神经网络中的参数，模型学习过程就是权重的优化过程，该优化使用随机梯度下降（SGD）算法来执行。

机器学习方法通常容易过拟合，即过度关注训练集中存在的某些特定特征，而忽略一些新特征和隐藏的特征。这意味着对训练集的误差估计过于乐观，因此需要一组独立的训练样本进行无偏估计，同时也要使用一个独立的验证集进行验证。因此将可用数据分为 3 部分：训练集、验证集和测试集。采取交叉验证对 Text-CNN 和 logistic 回归模型进行防过拟合处理，对 Text-CNN 进行参数的选择。

3 实验与评估

3.1 数据集及实验环境

实验收集了 Debian 漏洞跟踪报告中的 1400 个程序，其中 8% 含有内存损坏漏洞。该程序集覆盖了各种常用程序类型，如数据处理程序、游戏、小桌面程序等。实验中训练集和验证集共占数据集的 70%，测试集占数据集的 30%。实验环境包括硬件环境与软件环境，如表 1 所示，其中软件环境基于 Docker。

表 1 实验环境

实验环境	硬件	配置
硬件环境	CPU	2.5 GHz Intel Core i76700
	内存	8 GB 1600 MHz DDR3
	GPU	GTX 1080Ti
软件环境	Docker	17.12.0-ce
	Ubuntu	16.04.3LTS
	python	2.7.13
	Sklearn	0.19
	cudnn	5.1
	cuda	8.0
	keras	1.2.0
	Nvidiadriver	384

3.2 系统运行

从数据集中分别提取静态特征和动态特征。通过基于随机探测算法的静态特征提取方法得到程序的静态特征集，共 90 MB 数据。图 5 所示为从静态特征集中选取的一条较短的静态特征序列，该特征序列由程序名、函数调用序列、样本类型标记组成。通过基于信息聚合的动态特征提取方法得到动态特征集，共 18 GB 数据。图 6 所示为从动态特征集中选取的一条较短的动态特征序列，该特征序列由程序名、事件序列、样本类型标记组成。

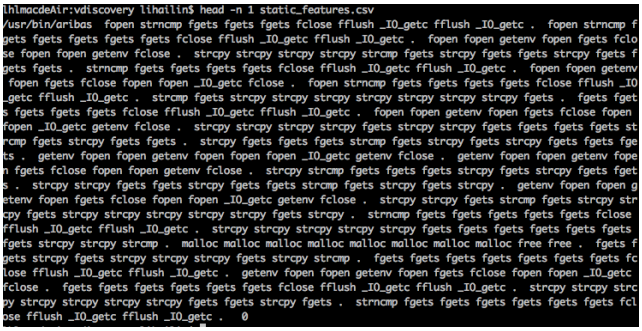

图 5 静态特征示例

图 6 动态特征示例

采用 tf-idf 算法对静态特征集和动态特征集进行向量化，处理为特征矩阵后参与机器学习模型的训练。

3.3 实验结果

为了对不同特征和分类方法进行有效评估，针对静态特征集或动态特征集，分别使用 logistic 回归模型、Text-CNN 与随机森林 3 种机器学习算法进行训练，得到 6 种分类器。采用 10 折交叉验证对分类器的检测结果进行评估，选取检测性能较好的分类器。

检测结果所用评估指标包括精确率、召回率与 *F*1 值。将带有内存损坏漏洞的样本以正样本（Positive）表示，将正常样本以负样本（Negative）表示，则在实际检测时会出现以下 4 种情况：

1）*TP*（True Positive）。表示正样本被正确检测为正样本的数量。

2）*FN*（False Negative）。表示正样本被误认为负样本的数量。

3）*TN*（True Negative）。表示负样本被正确分类为负样本的数量。

4）*FP*（False Positive）。表示负样本被误报为正样本的数量。

精确率（Precision）表示为 *TP*/(*TP*+*FP*)，即正确分类的正样本占所有被分类为正样本的比例；召回率（Recall）表示为 *TP*/(*TP*+*FN*)，即正确分类的正样本占所有正样本的比例。*F*1 值是精确率和召回率的调和均值，即 *F*1=2*PR*/(*P*+*R*)（*P* 代表精确率，*R* 代表召回率），相当于精确率和召回率的综合评价指标。评估结果如表 2 所示，并与通过随机抽样预测正样本的方法进行对比。因为正样本占所有样本数量的 8%，所以通过随机抽样预测正样本的精确率和召回率与正样本比例相等，都为 0.08。

表 2 评估结果

分类器	特征集	精确率	召回率	*F*1
Logistic 回归	静态	0.332	0.523	0.406
随机森林	静态	0.383	0.568	0.457
Text-CNN	静态	0.572	0.208	0.305
Logistic 回归	动态	0.342	0.432	0.381
随机森林	动态	0.401	0.543	0.461
Text-CNN	动态	0.421	0.552	0.478
随机预测	—	0.08	0.08	0.08

由表 2 可以看出，检测效果最好的分类器是对动态特征集采用 Text-CNN 模型训练所得到的分类器，精确率为 0.421，召回率为 0.552，*F*1 值为 0.478；对静态特征集采用 Text-CNN 模型训练所得到的分类器的召回率很低，原因可能是样本数据集的不均衡导致基于静态特征集采用 Text-CNN 模型训练所得的分类器在学习过程中产生了过拟合现象。

实验结果表明，本文方法对内存损坏类漏洞程序的检测精确率为 40%左右，召回率为 55% 左右，远高于随机预测性能，可以有效对内存损坏类漏洞进行漏洞检测。

4 结束语

本文利用随机探测算法对反编译后的程序进行静态特征提取，且在动态特征提取过程中通过定义参数子类型对参数进行信息聚合，得到信息较集中的动态特征序列。对提取的动态特征集和静态特征集分别运用 Text-CNN、Logistic 回归、随机森林等算法进行模型训练，从而实现了一种内存损坏类漏洞的专用检测方法。

由于轻量级特征提取方式不存在指令注入等复杂耗时的操作，不需要源码便可进行漏洞的预分类，因此本文提出的漏洞检测方法具备以下优点：

1）不需要源代码。特征通过二进制程序的动态执行与静态反汇编提取，允许在操作系统所带的二进制程序中进行漏洞检测。

2）全自动化。一些机器学习应用依赖于复杂的特征工程才能获得良好的性能，模型训练前需对大量候选特征进行筛选及预处理。本文检测方法能够自动从数据集中提取特征，直接用于模型训练。

3）可扩展性。本文检测方法仅利用了轻量级的动态特征提取技术与静态特征提取技术，消耗资源少、检测速度快，具有极高的可扩展性。

然而，数据集不均衡导致训练阶段可用正样本较少，对模型准确性具有一定影响。下一步工作可基于卷积神经网络等机器学习算法对分类器进行改进，以消除数据集不均衡对模型检测效果的影响。（责编 马珂）

参考文献：

[1] RAWAT S, MOUNIER L. Finding Buffer Overflow Inducing Loops in Binary Executables[C]// IEEE.IEEE Sixth International Conference on Software Security and Reliability, June 20−22, 2012, Gaithersburg, MD, USA.NJ: IEEE, 2012:177−186.

[2]SONG Yuanyuan, Sovarel A N, YANG Jing. Detection and Prevention of Memory Corruption Attacks[EB/OL].http://www.cs.virginia.edu/~jy8y/publications/cs77105.pdf, 2018−9−20.

[3] WANG Xiajing, HU Changzhen, MA Rui, et al. A Survey of the Key Technology of Binary Program Vulnerability Discovery[J]. Netinfo Security,2017,17(8):1−13.

王夏菁，胡昌振，马锐，等. 二进制程序漏洞挖掘关键技术研究综述 [J]. 信息网络安全，2017, 17(8): 1−13.

[4] VIEGA J, BLOCH J T, KOHNO Y, et al. ITS4: A static Vulnerability Scanner for C and C++ code[C]// IEEE. 16th Annual Computer Security Applications Conference (ACSAC'00), December 11−15, 2000, New Orleans, LA, USA.NJ:IEEE, 2000:257−267.

[5]Google Inc. and Cpplint Developers. Cpplint[EB/OL].http://sourceforge.net/ projects/cpplint/,2018 −4 −6.

[6]THENAULT S. Pylint— Code Analysis for Python[EB/OL]. https://www.pylint.org,2018−9−20.

[7] EVANS D, LAROCHELLE D. Improving Security Using Extensible Lightweight Static Analysis[J]. IEEE Software, 2002, 19(1):42−51.

[8] YAMAGUCHI F, GOLDE N, ARP D, et al. Modeling and Discovering Vulnerabilities with Code Property Graphs[C]// IEEE. 2014 IEEE Symposium on Security and Privacy, May 18−21, 2014, San Jose, CA, USA.NJ:IEEE, 2014:590−604.

[9] XU Youfu, WEN Weiping, WAN Zhengsu. Vulnerability−based Model Checking of Security Vulnerabilities Mining Method[J]. Netinfo Security, 2011, 11(8): 72−75.

徐有福 ; 文伟平 ; 万正苏 . 基于漏洞模型检测的安全漏洞挖掘方法研究 [J]. 信息网络安全 , 2011, 11(8): 72−75.

[10] CAI Jun, ZOU Peng, MA Jinxin, et al. SwordDTA: A Dynamic Taint Analysis Tool for Software Vulnerability Detection[J]. Journal of Wuhan University, 2016, 21(1):10−20.

[11] ZHANG Xiong, LI Zhoujun. Review of Fuzzy Testing Technology[J].Computer Science, 2016, 43(5):1−8.

张雄 , 李舟军 . 模糊测试技术研究综述 [J]. 计算机科学 , 2016, 43(5):1−8.

[12] CADAR C, DUNBAR D, ENGLER D. KLEE: Unassisted and Automatic Generation of High−coverage Tests for Complex Systems Programs[C]//ACM. The 8th USENIX Conference on Operating Systems Design and Implementation, December 8 − 10, 2008 , San Diego, California ,USA. New York:ACM,2009:209−224.

[13] WANG T, WEI Tao, GU Guofei, et al. TaintScope: A Checksum−Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection[C]// IEEE. 2010 IEEE Symposium on Security and Privacy, May 16−19 ,2010, Berkeley/Oakland, CA, USA.NJ:IEEE, 2010:497−512.

[14] SEN K, AGHA G. CUTE and jCUTE: Concolic Unit Testing and Explicit Path Model−Checking Tools[C]// ACM. The 18th International Conference on Computer Aided Verification, August 17 − 20, 2006, Seattle, WA,USA.New York:ACM, 2006:419−423.

[15] SANTOS I, DEVESA J, BREZO F, et al. OPEM: A Static−Dynamic Approach for Machine−Learning−Based Malware Detection[M]//Springer. International Joint Conference CISIS' 12−ICEUTE´12−SOCO´12 Special Sessions. Heidelberg :Springer Berlin Heidelberg, 2013:271−280.

[16] YAMAGUCHI F, LINDNER F, RIECK K. Vulnerability Extrapolation: Assisted Discovery of Vulnerabilities using Machine Learning[C]//ACM. The 5th USENIX Conference on Offensive Technologies , August 8−12, 2011, San Francisco, CA ,USA.New York:ACM,2012:13.

[17]Springer. Transactions on Rough Sets[M]. Heidelberg: Springer−Verlag Berlin Heidelberg, 2008.

[18] GRIECO G, GRINBLAT G L, UZAL L, et al. Toward Large−Scale Vulnerability Discovery Using Machine Learning[C]// ACM. The Sixth ACM Conference on Data and Application Security and Privacy , March 9 − 11, 2016, New Orleans, Louisiana, USA.New York:ACM, 2016:85−96.