人工智能技术在安全漏洞领域的应用

孙鸿宇[1,2]，何远[2]，王基策[2]，董颖[2]，朱立鹏[1,2]，王鹤[1,2]，张玉清[1,2]
（1. 西安电子科技大学网络与信息安全学院，陕西 西安 710071；2. 中国科学院大学国家计算机网络入侵防范中心，北京 101408）

摘 要：软件数量的大规模增长以及复杂性的增强给软件安全漏洞的研究带来了严峻的挑战，以人工的方式进行安全漏洞研究的效率较低，无法满足网络空间安全的需要。因此，如何将机器学习、自然语言处理等人工智能技术应用于安全漏洞的研究已成为新的热点，人工智能技术能够智能化地处理漏洞信息来辅助安全漏洞研究，同时提高安全漏洞挖掘的效率。首先分析了安全漏洞的自动化挖掘、自动化评估、自动化利用和自动化修补等关键技术，指出安全漏洞挖掘的自动化是人工智能在安全漏洞领域应用的重点，然后分析和归纳了近年来提出的将人工智能技术应用于安全漏洞研究的最新研究成果，指出了应用中的一些问题，给出了相应的解决方案，最后展望了安全漏洞智能研究的发展趋势。

关键词：漏洞挖掘；机器学习；人工智能
中图分类号：TP18
文献标识码：A
doi: 10.11959/j.issn.1000−436x.2018137

Application of artificial intelligence technology in the field of security vulnerability

SUN Hongyu[1,2], HE Yuan[2], WANG Jice[2], DONG Ying[2], ZHU Lipeng[1,2], WANG He[1,2], ZHANG Yuqing[1,2]
1. School of Cyber Engineering, Xidian University, Xi'an 710071, China
2. National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing 101408, China

Abstract: The large number of software and the enhancement of complexity have brought severe challenges to the research of software security vulnerabilities. The efficiency of manual research on security vulnerabilities is low and cannot meet the needs of cyberspace security. Therefore, how to apply artificial intelligence techniques such as machine learning and natural language processing to the study of security vulnerabilities has become a new hot spot. Artificial intelligence technology can intelligently process vulnerability information, which can assist in the research of security vulnerabilities and improve the efficiency of research on security vulnerabilities such as vulnerability mining. Firstly, the key technologies of automatic mining, automatic assessment, automatic exploitation and automatic repair of security vulnerabilities were analyzed, which pointed out that the automation of security vulnerability mining was the key of the application of artificial intelligence in the field of security vulnerability. Then, the latest research results of applying artificial intelligence technology to the research on security vulnerabilities was analyzed and summarized in recent years, which pointed out some problems in the application and gave corresponding solutions. Finally, the development trend of intelligent research on security vulnerabilities was prospected.

Key words: vulnerability mining, machine learning, artificial intelligence

收稿日期：2018−06−19；**修回日期**：2018−07−20
基金项目：国家重点研发计划基金资助项目（No.2016YFB0800700）；国家自然科学基金资助项目（No.61572460, No.61272481）；信息安全国家重点实验室开放课题基金资助项目（No.2017-ZD-01）；国家发改委信息安全专项基金资助项目（No.(2012)1424）
Foundation Items: The National Key Research and Development Program of China (No.2016YFB0800700), The National Natural Science Foundation of China (No.61572460, No.61272481), The Open Project Program of the State Key Laboratory of Information Security (No.2017-ZD-01), The National Information Security Special Project of National Development and Reform Commission of China (No.(2012)1424)

1 引言

安全漏洞[1]是指信息技术、信息产品、信息系统在需求、设计、实现、配置运行等过程中，有意无意之间产生的缺陷。这些缺陷以各种各样的形式存在于信息系统的各个层次、环节之中，对整个信息系统有着很重要的影响，且一旦被恶意主体利用，会影响构建在信息系统之上正常服务的运行，对信息系统的机密性、完整性以及可用性造成严重损害，因此对安全漏洞的研究是网络空间安全研究的核心内容之一。

安全漏洞的研究主要分为漏洞挖掘、漏洞分析及利用、漏洞评估、漏洞修复等。漏洞挖掘是指安全研究人员利用各种工具对软件、系统代码进行审计，对软件执行过程进行分析来查找缺陷的过程。漏洞的分析及利用是指对软件或系统中挖掘到的缺陷进一步的分析，确认该缺陷是否为安全漏洞，若为安全漏洞则进一步判断漏洞类型并开发概念验证性的攻击代码（PoC, proof of concept）。漏洞评估是指对研究人员提交的安全漏洞进行评估，分析该漏洞的危害性、影响范围等，并对漏洞修复提供指导意见。漏洞修复则会根据漏洞评估的危害等级来进行漏洞修复，优先修复危害等级高的漏洞。整个安全漏洞周期如图 1 所示。

近年来，随着软件系统的大规模增长和复杂性的增强，安全漏洞报告提交数量呈现逐渐增长趋势，美国国家漏洞数据库（NVD，national vulnerability database）披露的历年漏洞记录数量如图 2 所示。由图 2 可知，2017 年的漏洞记录数量是 2016 年全年的 2 倍多。软件的快速发展带来漏洞数量的增多，增加了计算机用户在使用网络服务时面临的安全风险，同时威胁性大的安全漏洞给网络空间安全带来了巨大的危害。例如，2017 年出现的“WannaCry”勒索攻击，WannaCry 利用微软服务器消息块（SMB, server message block）协议的远程溢出漏洞，并搭载美国国家安全局（NSA, national security agency）制造“永恒之蓝（EternalBlue）”网络武器，导致攻击威力倍增，在短短数小时内就发动数万次攻击，受害国家超过 150 个，政府、企业、医疗、高校等各行业均有 IT 设备中招，波及大量公司，形成了全球性互联网灾难。面对安全漏洞带来的这些严峻的挑战，如何实现自动化漏洞挖掘、自动化漏洞利用生成以及自动化漏洞修补等都是当前急需解决的课题。

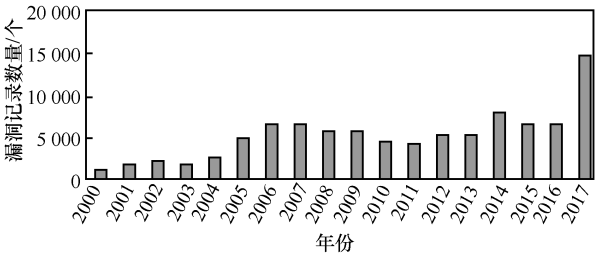


图 2 美国国家漏洞数据库（NVD）披露的历年漏洞记录数量

近年来，人工智能（AI, artificial intelligence）技术有了较大的发展，利用人工智能技术可以对漏洞报告以及程序代码自动化处理并提取有效的信息，以此来实现安全漏洞的自动化研究。将人工智能技术应用于网络安全领域主要是利用机器学习（ML, machine learning）技术以及自然语言处理（NLP, natural language processing）技术等来进行安全漏洞研究。

机器学习[2]是指研究计算机模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善计算机系统自身的性能，其根本目的是为了实现人工智能。近年来，机器学习的迅速发展，使各个行业开始考虑将机器学习算法应用到各个研究方向中，例如，机器学习应用于统计学习[3]、模式识别[4]、数据挖掘[5]等方面均取得了成功。

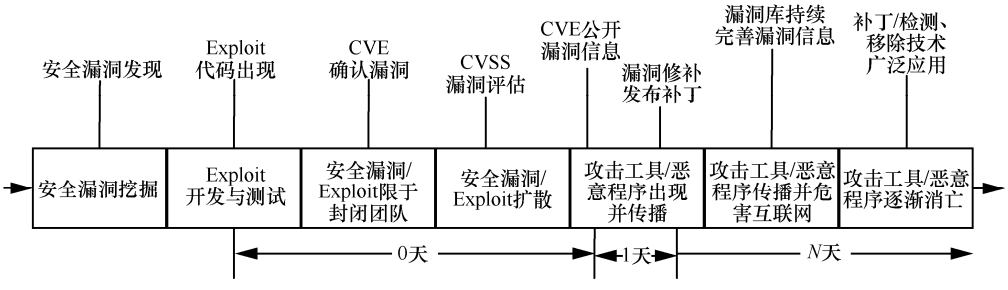


图 1 安全漏洞周期

机器学习算法根据学习方法的不同，可以分为监督学习（supervised learning）、非监督学习（unsupervised learning）和半监督学习（semi-supervised learning）。监督学习是指通过已有的一部分输入数据特征与输出数据标签之间的相应关系，生成一个函数，将输入映射到相应的输出，这种学习方式在多数条件下应用于分类计算。非监督学习是指直接对输入数据进行建模，比如聚类算法。半监督学习是指综合利用有标签的数据和无标签的数据来生成合适的分类函数。机器学习能够让研究人员脱离大量繁杂重复的工作，因此成为实现自动化的最佳选择之一。

深度学习（DL, deep learning）[6]是由多个非线性特征变换构成处理层来对数据进行表征学习,是机器学习算法的最新研究成果，其目的是通过经验和数据改进计算机系统，实现人工智能。深度学习算法能够提高“浅层”机器学习算法的性能，同时扩大了机器学习的应用范围，如将深度学习算法应用于计算机视觉[7-8]、自然语言处理[9]等方面均取得了巨大成功。

自然语言处理是研究人与计算机之间用自然语言进行有效通信的理论和方法，其最终目的是实现人机之间的自然语言通信，是人工智能技术的一部分。自然语言处理能够自动化提取本文内容，比如自然语言处理能够对漏洞报告、安全网站的相关信息进行提取，并进行实体识别以及提取实体之间的关系，同时也能够根据训练结果进行自然语言生成，自然语言处理提出的各种语义模型对文本数据进行处理具有很大的优势。自然语言处理在语音合成与识别、机器翻译、人机对话等方面产生了深远的影响。

在安全领域，研究人员将机器学习算法引入恶意软件检测[10]以及入侵检测[11]，获得了巨大的成功。更多的研究人员考虑将机器学习应用于安全漏洞的研究中，实现漏洞挖掘等研究的智能化。将机器学习应用于漏洞挖掘一般采用监督学习方式，其过程可以分为数据预处理、建立机器学习算法模型、对机器学习模型进行训练、测试以及评估等过程。利用文本挖掘技术对程序源代码进行预处理，并采用机器学习进行漏洞挖掘的过程如图 3 所示。具体而言，首先采用文本挖掘技术对源代码的信息进行预处理，该步骤包括过滤注释、提取源代码的语义信息、将语义信息映射到向量空间等过程，然后采用诸如决策树（DT, decision tree）、支持向量机（SVM, support vector machine）等机器学习算法来构造漏洞挖掘模型并进行训练以及测试，最后对构建的模型进行评估，从而实现漏洞挖掘。

2　常见的漏洞研究方法

2.1　漏洞挖掘

对安全漏洞进行挖掘一直是网络空间安全的重点内容。安全漏洞挖掘是指利用各种工具、技术，寻找计算机系统中存在的漏洞。根据是否运行程序代码，漏洞挖掘技术可以分为静态分析技术和动态分析技术。

2.1.1　静态分析技术

静态分析技术是指在不运行程序的情况下，对

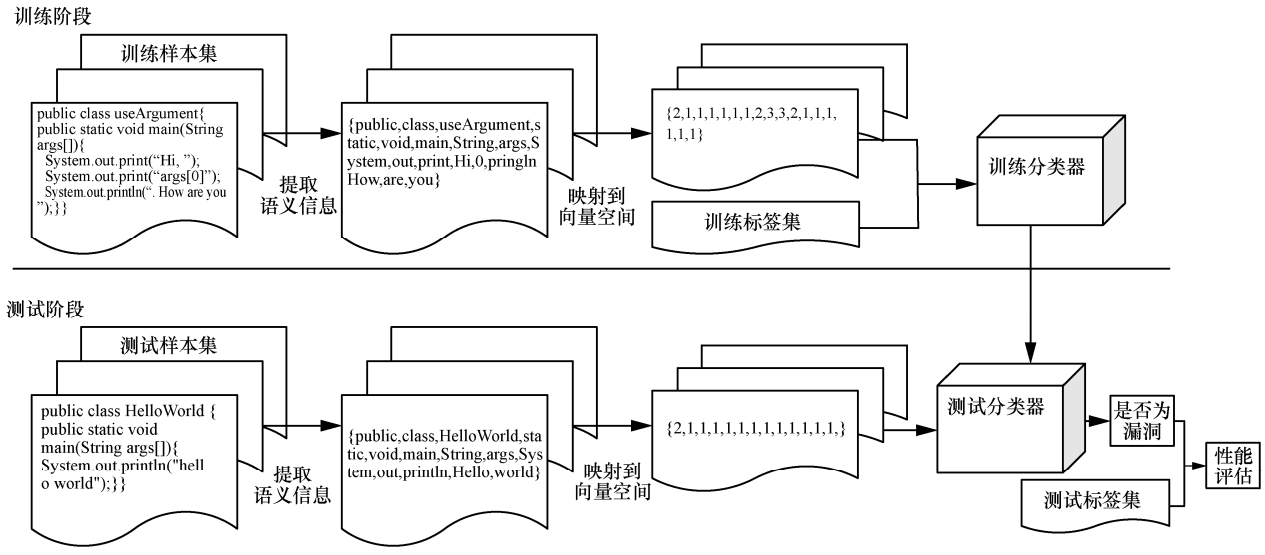


图 3　文本挖掘与机器学习结合应用于漏洞挖掘的过程

目标程序进行分析以及检测，从而发现目标程序中可能包含的安全漏洞。静态分析技术主要包括源代码扫描、静态污点分析、可达路径分析、静态符号执行等技术。

源代码扫描通过检测程序中不符合安全规则的文件结构、命名规则、堆栈指针等信息来发现可能隐含的安全缺陷，并需要预先定义出相应的漏洞模式，然后进行源代码扫描，该方法仅能够对某种漏洞进行检测。静态污点分析[12]通过分析源码或字节码中语句或指令之间的静态依赖关系来判断污点标记所有可能的传播途径，并以此进行来漏洞挖掘，其往往需要较大的空间开销，误报率高，自动化程度不够高。可达路径分析根据程序执行方向将代码解析为一个有向图，通过对图连通性质的研究并结合约束求解实现对程序的分析，约束求解的困难性一直是可达路径分析技术的一个关键问题。静态符号执行[13]采用抽象符号代替程序变量，并模拟程序执行，它能够在复杂的数据依赖关系中发现变量之间本质的约束关系，静态符号执行面临着路径执行空间爆炸、很难处理循环或递归等问题，同时对硬件计算能力要求较高，这些问题制约着静态分析技术的发展，无法进行大规模的自动化漏洞挖掘。

除此之外，静态分析技术还包括二进制文件比对技术[14]、手工测试技术[15]等。

静态分析技术能够高效快速地完成对程序代码的检查，并且其代码覆盖率较高、漏报较少。然而，由于静态分析缺乏运行时的数据，并且缺乏动态的测试过程和细粒度的安全评估，导致静态分析技术准确率较低、误报较多。

2.1.2 动态分析技术

动态分析技术通过观察程序运行过程中的运行状态、寄存器状态的异常来发现漏洞。动态分析技术主要包括 Fuzzing 测试、动态污点分析技术、动态符号执行技术等。

Fuzzing 测试[16]通过生成大量畸形测试数据来测试程序的顽健性和安全性，其核心是测试用例生成技术。Fuzzing 测试是漏洞挖掘最有效、最多产的方法。动态污点分析技术[17]对程序的污点数据在系统程序中的传播进行实时监控，设计有效的污点传播逻辑来保证污点分析精确度。动态符号执行技术[18]通过执行相应的测试程序，对输入有关的变量进行符号化，而对那些与输入无关的变量只采取实际执行，动态符号执行具有代码覆盖率高、准确性高的特点，能够分析更大规模的程序。

动态分析技术具有较高的漏洞挖掘准确率，但动态分析技术代码覆盖率相对较低，条件不满足时代码将无法执行，同时存在漏报问题。考虑到静态测试可以覆盖所有代码，所以目前结合静态和动态分析进行漏洞挖掘成为主流的安全漏洞研究方式。

2.2 漏洞利用

漏洞利用是获得系统控制权限的重要途径。具体而言，漏洞利用是通过一段带有攻击载荷（Payload）的程序触发某个漏洞（或几个漏洞），越过具有漏洞的程序限制，进而控制目标系统中代码的运行过程，即执行攻击者执行代码的过程。对漏洞利用的研究就是寻找已确认漏洞的可利用点，然而大多数漏洞尚未找出相应的漏洞可利用点。

Exploit-DB 对漏洞利用进行了收集。Exploit-DB 是与公共漏洞和披露（CVE, common vulnerabilities and exposures）[19]兼容的包含已公开的部分软件漏洞的漏洞利用的数据库，Exploit-DB 存储了大量的漏洞利用程序，并且包含最新漏洞的相关信息，供渗透测试人员和漏洞研究人员使用。

2.3 漏洞评估

安全漏洞威胁严重性评估是漏洞研究的重要组成部分，它依据漏洞自身相关属性通过综合衡量相关漏洞威胁评估指标来获得安全漏洞的威胁严重性程度，本文也称这个过程为漏洞分级。漏洞分级方法分为定性和定量 2 种。常见的定性方式是将漏洞评估为高、中、低共 3 档，也有如微软安全公告评级系统将漏洞分为紧急、重要、中、低共 4 档。常用的漏洞定量系统是通用漏洞评分系统（CVSS, common vulnerability scoring system）[20]。

CVSS 是一个面向广大安全厂商用于安全漏洞威胁严重等级评估的、公开免费的定量风险评估系统。CVSS 评分过程包括 3 种评分项：基本指标、临时指标、环境指标，每一种指标又包含了一组不同的度量以及评分标准。最后对相应的评分结果进行加权处理，从而获取一个用来衡量安全漏洞威胁严重程度的分值。一般而言，CVSS 评分在 7~10 表示严重漏洞，4~6.9 表示中等漏洞，0~3.9 表示低级漏洞，根据这些评分结果优先对严重性漏洞进行修

复处理。

2.4　漏洞修补

对漏洞进行修复的过程是从收到漏洞报告的通知开始的，根据漏洞的可利用性、漏洞风险和可用的知识等信息进行漏洞修复，共产生了 3 种修复方案。

1) 如果已知并记录了该漏洞的类型，开发人员将继续分析与该漏洞相关的代码，设计并实现一个解决方案，然后使用识别该漏洞的技术进行测试直至安全漏洞完成修复。

2) 如果漏洞类型是已知的并且由核心安全团队记录，但之前并没有遇到这样的安全漏洞，那么就由开发团队和核心安全团队合作分析识别该漏洞并设计相应的解决方案。

3) 如果是未知漏洞类型，那么核心安全团队将与来自不同领域的专家和开发人员进行协作，以开发针对该漏洞的通用解决方案。

此外，一个通用的解决方案还需要考虑不同的产品领域、不同的使用技术和不同的编程语言。而且安全专家还会与框架设计专家协作，以实现开发人员可以使用的库来避免一些安全漏洞。同时开发指导方针，使开发人员能够使用这些指导方针来解决这类漏洞。就目前技术手段而言，大多数情况下依然采用人工的方式进行漏洞修复。

3　机器学习应用于漏洞研究

常见的安全漏洞研究方法往往要求安全研究人员具备足够多的专业性知识，这种漏洞研究方法的效率很低。将人工智能技术应用于安全漏洞来实现安全漏洞的智能化研究，能够提升安全防护的效率。自 2007 年以来，从 IEEE、ACM、Springer 等数据库收录的将人工智能技术应用于安全漏洞研究方面的文献情况如图 4 所示。由图 4 可知，漏洞挖掘依然是安全研究人员关注的重点，而其他方面的研究尚少。因此本文重点侧重于漏洞挖掘方面的研究。

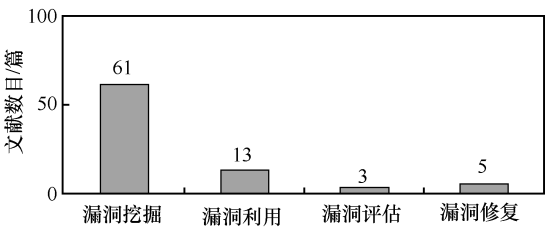


图 4　机器学习用于漏洞研究方面文献情况

3.1　自动化漏洞挖掘

将机器学习应用于漏洞挖掘一直受到安全研究人员的关注，其本质是将漏洞挖掘问题视为程序分类问题或聚类问题，将包含漏洞的程序从正常程序中区分出来或将包含漏洞的程序聚集在一起。近年来，安全研究人员通过对漏洞产生原理、漏洞产生条件等进行深入研究，采用各种学习算法构建了不同的漏洞挖掘模型进行漏洞挖掘。自 2007 年以来，将机器学习应用于漏洞挖掘方面的文献变化情况如图 5 所示。众所周知，程序源代码具有丰富的特征信息，比如，抽象语法树（AST, abstract syntax trees）、应用程序接口（API, application programming interface）调用等信息，这些信息均用离散的符号进行表示，无法直接作为机器学习算法的输入，因此还需要对程序代码进行处理。根据处理方式的不同，所得到的特征信息也不同，由此本文将漏洞挖掘模型分为基于软件度量的漏洞挖掘模型和基于语法语义特征的漏洞挖掘模型。除此之外，对漏洞挖掘模型进行评估又有 2 种评估方式：一种是能否在现实应用程序中挖掘到漏洞；另一种则是通过各种指标来评估漏洞模型，根据分类预测的结果，即真正的正样本（TP, true positive）、误报率（FP, false positive）、真正的负样本（TN, true negative）以及漏报率（FN, false negative）来计算精确率（precision）、召回率（recall）、准确率（accuracy）等指标，计算方式如下。

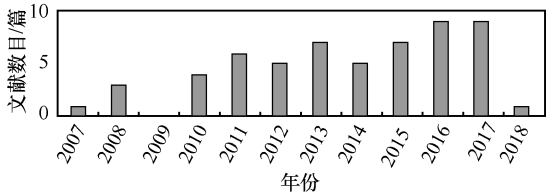


图 5　机器学习应用于漏洞挖掘文献变化情况

$$precision = \frac{TP}{TP + FP} \tag{1}$$

$$recall = \frac{TP}{TP + FN} \tag{2}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

3.1.1　基于软件度量的漏洞挖掘模型

软件度量是对软件开发项目、过程及其产品进行数据定义、收集以及分析的持续性定量化过程，目的在于对开发项目加以理解、预测、评估、控制和改善。软件度量是对软件特定实体属性的量化表

示，能够提供软件的各种信息，同时又可以通过软件工具获取，因此软件度量成为安全研究人员进行漏洞挖掘的特征选择之一。常用的软件度量包括复杂度（complexity）度量、代码变化（code churn）度量、耦合度（coupling）、内聚度（cohesion）、开发者活动（developer activity）度量等度量指标。文献[21-81]在不同程度上构建了基于软件度量的漏洞挖掘模型，本文则对其进行了分析与总结，具体如图 6 所示。

早期的软件度量是由文献[21-22]提出的，其目的是用来检测程序故障而并非进行漏洞挖掘。复杂度度量是安全研究人员的优先选择，正是因为软件的高复杂度阻碍了研究人员对程序的理解，因此越是复杂的软件越晦涩难懂，越容易产生漏洞。本文对比了文献[23-33]的漏洞挖掘模型性能，发现围绕着 Mozilla Firefox 进行漏洞挖掘的模型性能均较差，并且复杂度度量产生的影响会因为项目的变化而变化，同时相关性分析表明安全漏洞与软件复杂度之间的相关性较弱，这也印证了复杂度度量召回率低的实验结果。代码变化度量定义为软件版本与版本之间代码行数的变化情况。代码的变化直接与软件缺陷相关，从而很可能产生安全漏洞，因此研究人员采用该度量进行漏洞挖掘。文献[23-24,31,35]均采用了代码变化度量进行漏洞挖掘，研究结果表明，代码变化度量几乎和复杂度度量具有相似的实验结果，相关性分析表明了代码变化度量和漏洞之间的相关性并不强烈，然而文献[37-39]进一步扩大了代码变化度量的范围，考虑了开发者经验情况、每次更改花费时间等，利用代码提交（VCC, vulnerability-contributing commit）挖掘可疑的漏洞，并取得了很好的实验效果，然而这些软件度量与安全漏洞之间的弱相关性依然制约着这些软件度量的有效性。在计算机设计中，耦合度和内聚度是衡量模块独立程度的标准。耦合度是指在程序中，模块与模块之间信息或参数依赖的程度。内聚度是指功能相关的程序组合成一模块的程度。一般而言，高内聚性与软件的顽健性、可靠度等相关，而低内聚性也代表不易维护、不易测试。因此，将耦合度和内聚度相结合为挖掘漏洞提供了可能，文献[29-30]将复杂度、耦合度以及内聚度结合进行漏洞挖掘，实验结果表明，低耦合高内聚的文件往往不容易产生漏洞，相反，高耦合低内聚的文件更容易产生漏洞，这也表明了耦合度和内聚度在定义上的正确性。文献[31-32]对开发者活动度量进行了分析，开发者活动度量关注核心人员是否对源代码进行修改、单人负责还是多人共同负责等。他们的研究表明，多个不同的开发人员对源代码文件更改比一个开发人员对源代码文件更改更容易受到攻击。文献[34]研究了软件度量与软件漏洞之间的相关性。除此之外，文献[36,41-44]从其他方面比如上下文信息、漏洞库相关信息进行漏洞挖掘，文献[43]利用上下文信息提高了 XSS 漏洞挖掘效率，文献[44]则表明了难以利用漏洞库的相关信息进行有效的漏洞挖掘。

总体来看，选择软件度量进行漏洞挖掘依然不能满足研究需求，虽然如耦合度、内聚度、开发者活动等度量在一定程度上能够反映哪些文件可能会包含漏洞，但其构建的漏洞挖掘模型的性能依然较低。这表明这些软件度量并不适合用于漏洞挖掘，同时漏洞挖掘比故障检测要复杂得多。软件度量是对软件性质及其规格的测量，能够对软件整体的性质进行量化，然而这些性质在本质上和安全漏洞本身并没有很强的相关性。因此，利用软件度量进行漏洞挖掘并不合适，必须从漏洞本身出发，开发结合安全漏洞的代码特征，才能更好地将机器学

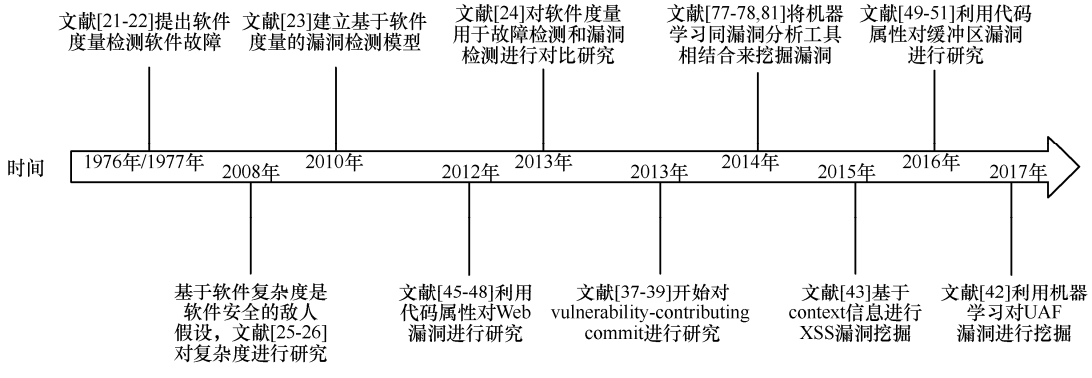


图 6 基于软件度量的漏洞挖掘模型历程

习应用于漏洞挖掘。

代码属性不同于软件度量，它是对软件度量的进一步发展，但并不是对软件整体信息的概括，而是需要结合具体的某类漏洞的知识，要求研究人员对该类漏洞的产生原理有着充分的了解，对该类型漏洞利用过程中的相关信息有深入的研究，并且能够从代码级别对这些信息进行统计，以这些信息为特征进行漏洞挖掘。代码属性是从安全漏洞的相关信息出发的，并且能够在代码层次上进行统计，将程序代码与安全漏洞连接起来，从而能够取得较好的检测效果，但代码属性的确定涉及专业的知识领域，需要专家经验来确定相应的特征选择。

文献[45-48]首先利用代码属性对 Web 漏洞进行研究，提出了一系列与漏洞相关的信息，比如采取 Sanitization 处理的节点数量等特征信息。这些信息能够从代码上进行统计，利用这些信息来挖掘漏洞，均取得了较好的实验结果。文献[49-53]利用代码属性对缓冲区溢出漏洞进行挖掘，从 Sink 分类、输入分类、输入验证等方面定义了一系列与缓冲区溢出漏洞有关的属性，并开发了能够根据代码自动化对这些信息进行统计的工具，他们的实验也获得不错的检测性能。这表明代码属性是漏洞挖掘的有效特征。但目前关于代码属性的研究主要集中在 Web 漏洞以及缓冲区溢出漏洞等方面，能否开发出新的代码属性来提高已有的漏洞挖掘模型性能还需要深入的探索。同时代码属性的应用范围太窄，不同类型的漏洞对代码属性的要求也不尽相同，能否找出适合挖掘其他类型漏洞的代码属性还需要研究人员进行持续的研究。最后，代码属性均是根据专家经验进行选择的，这些特征往往会带有一定的主观性，这种主观性会影响机器学习模型的效果。因此，本文可以通过让多个专家来定义自己认为重要的特征，然后从这些特征中选取能够有效提高效率的特征来缓解这种情况，然而这将带来更加繁重的工作。事实上，本文是希望减少甚至尽可能消除对专家经验的依赖性，因此需要客观地、自动化地对特征进行选择并实现漏洞挖掘，从而让专家从手工定义漏洞探测特性的烦琐工作中解脱出来。在软件度量方面实现漏洞特征的自动化选择是一个长期的过程，现阶段还需要依赖专家经验提升漏洞挖掘模型的性能。表 1 展示了部分基于软件度量的漏洞挖掘模型的性能。

表 1　部分基于软件度量的漏洞挖掘模型的性能

文献	性能			特征选择
	准确率	召回率	精确率	
文献[23]		20%	66.7%	复杂度度量
文献[25]	>90%			复杂度度量
文献[29]	72.85%	74.22%	4%	复杂度、耦合度以及内聚度
文献[43]	92.6%	88%	93.4%	上下文信息
文献[48]		77%	72%	代码属性
文献[50]	92.1%	95%	80.9%	代码属性

3.1.2　基于语法语义特征的漏洞挖掘模型

基于语法语义特征的漏洞挖掘模型又可以具体分为基于语义的漏洞挖掘模型和基于语法的漏洞挖掘模型。文献[54-87]表述了基于语法语义特征的漏洞挖掘模型的变化，其发展历程如图 7 所示。

基于语义的漏洞挖掘模型，主要是利用文本挖掘技术来获取程序源代码中的语义信息。文本挖掘是指从文本文件中提取有价值的知识，并且利用这些知识更好地组织信息的过程。将文本挖掘应用于漏洞挖掘的研究主要有 2 个方面。首先是通过对程序开发文档或程序注释进行分析并挖掘可能存在的漏洞。据本文统计，尚未有这方面的研究，文献[54]利用自然语言技术，能够有效地进行漏洞挖掘，能否

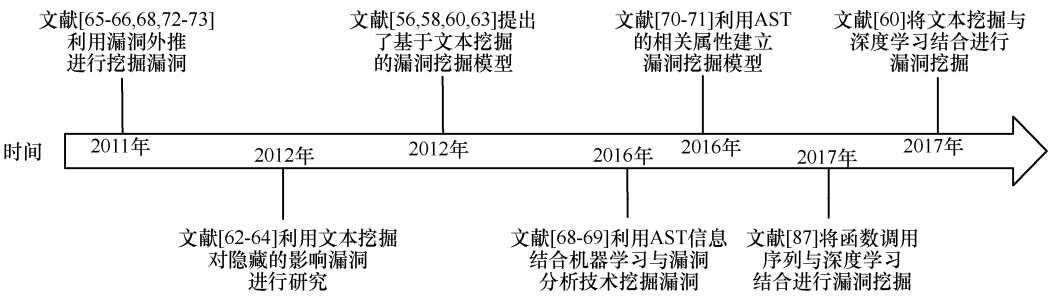


图 7　基于语法语义特征的漏洞挖掘模型历程

利用自然语言处理技术来处理开发文档或注释并以此进行漏洞挖掘值得关注。其次是对源代码进行文本挖掘，提取源代码的有效信息来挖掘漏洞。表 2 给出了部分基于语法语义特征的挖掘模型的性能。

表 2 部分基于语法语义特征的漏洞挖掘模型的性能

文献	性能			处理方式
	准确率	召回率	精确率	
文献[55]	87%	88%	85%	一元语义模型
文献[57]	92.25%	87.21%	95.78%	二元语议模型
文献[58]	75.98%	64.78%	58.75%	一元语义模型
文献[60]	92.87%	90.17%	94.71%	深度学习+文本信息

文献[55-60]等采用文本挖掘与机器学习结合的方法来挖掘漏洞，采用 *N*-gram 以及统计词频对源代码进行表征，使这些模型均获得了较高的检测率。*N*-gram 语义模型能够记录上下文中相邻词间的搭配信息，可以通过 *N* 个词语出现的概率来推断语句的结构。一般而言，*N* 越大，越能提供更好的语义效果，文献[57]采用 Bigram，比文献[55,58]采用的 Unigram（1-gram）取得了更好的模型效果。然而当 *N*>3 时会引起特征爆炸，加重机器学习对数据处理的负担，限制了漏洞挖掘模型的性能，因此，文献[61]希望利用降维技术来提高基于软件度量以及语义特征的漏洞模型的性能，其实验结果表明项目内采取降维技术对模型并没有太大的增益。然而他们仅采用统计词频作为语义特征进行降维处理，并没有考虑语义的上下文信息，因此将 *N*-gram 语义模型与降维处理相结合能否提高模型效果还需要继续验证。事实上，仅依赖词频统计及 *N*-gram 语义模型漏洞挖掘模型仅对程序源代码进行了粗略的语义信息提取，缺少对代码语义信息进行深度提炼，同时引入了一些不必要的代码元素，降低了模型的有效性。Word2Vec 语义模型是近年来自然语言处理中常用的新型语义模型，该模型能够将单词映射到一个连续的实值向量，从而方便对自然语言进行数字化处理，能够自动实现单词语义相似性的对比，这为代码相似性计算提供了新思路。但 Word2Vec 语义模型应用于漏洞挖掘的研究较少。同时，文献[62-64]利用文本挖掘对隐藏的影响漏洞进行研究。

基于语法的漏洞挖掘模型主要是利用 AST 来表征程序语法进行安全漏洞挖掘，AST 是源代码的抽象语法结构的树状表现形式，树上的每个节点都表示源代码中的一种结构。文献[65-71]均从程序的 AST 相关信息进行安全漏洞挖掘的研究。文献[65]研究了 AST 的结构模式信息，结合机器学习算法对函数进行安全漏洞挖掘，并发现了一些漏洞。文献[66]研究了 AST 的一些相关信息（如变量、条件等信息）辅助代码审计，也发现了一些安全漏洞。文献[68-69]将 AST 与程序分析技术相结合进行漏洞挖掘，提高了漏洞挖掘准确率。文献[67]将程序的抽象语法树、控制流图、数据依赖图相结合，形成的代码属性图能更好地表征程序的结构信息。对代码属性图按规则遍历实现了漏洞的自动化挖掘，并取得了较好的效果，但规则的编写对安全专家有较高的依赖，能否将机器学习与代码属性图相结合进行漏洞挖掘是值得探讨的课题。文献[70-71]则从 AST 的相关属性进行漏洞挖掘，能够在一定程度上取得不错的成果。

除此之外，本文发现文献[65-66,68,72-73]等利用漏洞外推的概念进行漏洞挖掘。漏洞外推是指从已知漏洞的使用模式出发，利用这些模式来指导代码审计并识别具有类似模式的程序。这些文献从 API 的调用模式、AST 的结构模式、数据的传播模式等出发，结合机器学习来进行漏洞挖掘，均发现了一些未知的漏洞，这表明了他们方法的有效性。然而利用漏洞外推进行漏洞挖掘往往需要专业安全人员对已知漏洞进行深入研究，确定使用模型往往需要对某类型的漏洞进行深入分析，同时每种漏洞外推方式仅适合某一种模式，并不能检测到其他的漏洞，有较大的局限性。

近年来基于软件度量的挖掘模型以及基于语法语义特征的漏洞挖掘模型的文献的变化情况如图 8 所示。不少安全研究人员对二者的性能优劣进行了研究。Walden 等[74]和 Tang 等[75]在一个包含 223 个漏洞的手工数据集上进行研究并比较 2 种模型的性能。Walden 的实验结果表明基于语义特征的模型有更高的召回率和精确度。然而 Tang 认为 Walden 做得不够全面，且没有考虑单个组件的大小。Tang 的实验结果表明，基于软件度量的模型可以与基于语义特征模型相媲美。对于研究人员而言，软件度量是漏洞挖掘的实际选择，因为这些软件度量体系更完善，同时通过软件工具更容易获取、成本更低。本文认为无论是语法语义特征还是软件度量，均是对源代码信息的一种表征方式，将语法语义模型和

软件度量模型结合起来进行漏洞挖掘能够更好地提高漏洞挖掘效果。事实上，Zhang 等[76]提出了一种两层的复合漏洞挖掘模型，结合软件度量以及语义特征在相同的数据集上进行漏洞挖掘，其实验结果明显优于 Walden 的模型性能。这表明了语义模型和软件度量模型相结合的确能够提高模型的挖掘效果。但他们的方法是分散地进行语义特征以及软件度量的漏洞挖掘的，并没有将软件度量以及语义特征相融合进行漏洞挖掘。如何融合 2 种特征行漏洞挖掘依然是一个研究难点。

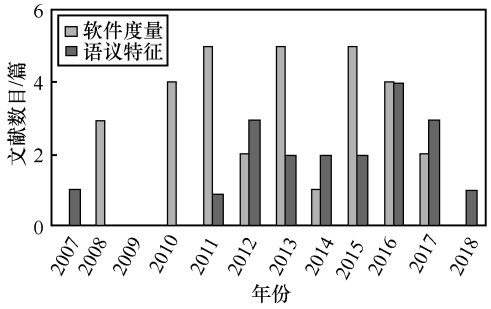


图 8　软件度量模型与语义特征模型文献对比

3.1.3　机器学习与程序分析技术相结合

文献[77-81]将漏洞挖掘模型与程序分析技术相结合来提高程序分析技术的性能。静态分析技术和动态分析技术在常见的漏洞挖掘方式中起到了非常大的作用，然而无论是静态分析还是动态分析技术均存在着相应的缺陷。利用机器学习来减缓或消除这些缺陷、提高程序分析技术的性能则是一个非常好的研究方向。

静态污点分析技术往往需要较大的空间开销，误报率高，机器学习能够快速处理大量的样本，结合机器学习来减低误报率则是一种可行的方法。符号执行的一个关键问题是路径执行空间爆炸问题，本文通过机器学习确定可疑函数集合，利用可疑函数集合来指导符号执行能够有效减少路径数量，减缓路径执行空间爆炸问题，提高了符号执行的性能。Fuzzing 测试需生成更有效的测试样例才能有效地触发漏洞，结合机器学习能够提高 Fuzzing 测试的效果。同时利用人工智能技术能够有效地提升代码覆盖率，进而有效地进行漏洞挖掘。由此可见，机器学习不仅能够用于挖掘漏洞，而且其分类结果也可以用来指导程序分析技术进行漏洞挖掘，提升漏洞挖掘效率。将机器学习同程序分析技术结合来解决静态分析高误报、低准确率、高动态分析漏报率、低代码覆盖率等问题，这为缓解空间开销大、约束求解难、路径执行空间爆炸等问题提供了新的思路，将机器学习同静态分析技术、动态分析技术结合进行漏洞挖掘也是一个可探讨的方向。

3.1.4　采用算法比较

在自动化漏洞挖掘过程中，机器学习算法在构建漏洞挖掘模型过程起到很重要的作用，漏洞挖掘文献中采用的机器学习算法对比情况，如图 9 所示。由图 9 可知，朴素贝叶斯（NB, naïve Bayes）、支持向量机（SVM, support vector machine）、逻辑回归（LR, logistic regression）、决策树（DT, decision tree）、随机森林（RF, random forests）是主要的漏洞挖掘算法。事实上，不同机器学习算法的性能是否存在显著性差异是一个争议性问题，文献[82-83]发现在 PROMISE 库的 10 个开源项目数据集上，不同的分类算法之间的确存在明显的差异，而文献[84]进一步分析了分类方法中参数的优化对缺陷预测性能的影响，结果表明这种影响不可忽略。将机器学习应用于漏洞挖掘，无法直接将源代码作为机器学习的输入，因此对源代码处理方式不同会产生不同的效果，同时不同的机器学习算法对数据特征信息的利用程度是不同的，也会产生不同的效果。因此必须采用多种机器学习算法对数据特征进行评估。在本文统计过程中发现，除了文献[69,78]采用了 10 种机器学习算法对选择的特征进行了性能对比之外，其他的文献仅仅采用了 3~4 种机器学习算法进行研究。我们认为先前的研究或许并没有取得特征数据的最佳检测效果，本文建议在未来的研究中，尽可能选择多种机器学习算法对数据特征进行建模并进行性能对比。

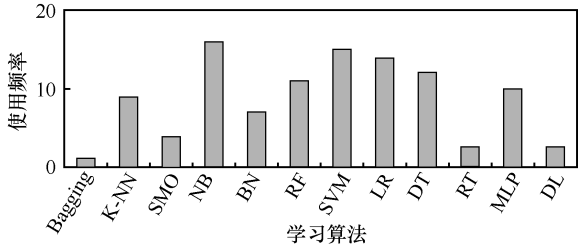


图 9　漏洞挖掘模型的机器学习算法对比

3.1.5　深度学习应用于漏洞挖掘

基于深度学习在图像识别、恶意软件检测等方面比其他“浅层”机器学习算法均能够获得更好的性能的经验性的证明，不少研究人员尝试着将深度学习引入漏洞挖掘领域。将深度学习应用于漏洞挖掘有 2 个方面的应用，一方面是利用深度学习模型

进行漏洞特征的自动化选择，这方面应用可以将深度学习与语法语义特征结合进行漏洞挖掘，另一方面是利用深度学习进行漏洞挖掘，这方面需要考虑：1) 如何将程序表征为适合深度学习模型的向量表示。应用程序具有丰富的特征，比如 AST、函数调用等，这些特征无法直接作为深度学习模型的输入，因此需要将这些特征转化为适合深度学习模型的向量表现形式；2) 漏洞挖掘的粒度。不同的特征信息具有不同的粒度，同时漏洞挖掘的粒度与漏洞定位有关，细粒度的漏洞挖掘能够更好地定位漏洞；3) 能否挖掘多种类型的漏洞。不同种类的安全漏洞对安全研究人员的要求也不相同，专业的安全研究人员往往针对某类安全漏洞进行深度挖掘，采用深度学习算法能否同时挖掘多种漏洞是一个非常有趣的问题；4) 如何选择深度学习模型。现有的深度学习模型有很多种，如何构建合适的深度学习模型来获取最佳的漏洞挖掘性能也是一个问题。

文献[85]将深度学习应用于程序分析，证实了深度学习应用于分析程序的可行性。而文献[59]利用深度学习辅助漏洞挖掘，采用长短期记忆（LSTM, long short-term memory）网络对文本特征进行自动化特征选择，以克服专家经验的主观性，获得了不错的性能。文献[86]从 API 调用以及库调用出发，提出了“Code Gadget”，这是一组语义上相互关联，但不一定连续的代码行。这种表征方式能够同时兼顾语义相关性并从细粒度上进行漏洞挖掘。细粒度的漏洞挖掘能够识别漏洞位置，以前的研究中很少对漏洞进行定位。文献[60]从 Token 级数据入手，采用文本挖掘与深度学习相结合的方式对软件组件进行漏洞挖掘，获得了比其他文本挖掘与机器学习结合更好的性能（性能比较如表 2 所示）。文献[87]从函数级数据出发，以函数调用序列为特征对比了不同的深度学习模型以及多层感知器（MLP, multi-layer perceptron）的性能，其实验结果发现采用深度学习模型的实验结果比同一数据集上采用 MLP 的实验结果要好得多。

就目前而言，将深度学习应用于漏洞挖掘还处于初步阶段。首先，文献[60,87]均表明了深度模型能够获得比“浅层”更好的研究性能，将软件度量或语法语义特征等与深度学习相结合，能否提高挖掘效果的研究还很少，这方面的研究依然有待探索。其次，细粒度的漏洞挖掘能够识别漏洞位置，这无疑扩展了漏洞挖掘的能力。在保证漏洞挖掘效果的前提下进行漏洞定位也是未来的一个研究方向。另外，文献[86]采用深度学习模型对 2 种漏洞进行挖掘，实验结果表明了深度学习能够对 2 种漏洞同时进行挖掘，但这种能力是否存在上限并没有给出明确的答案。最后，需要对不同的深度学习模型进行对比。相同的数据集上采用不同的神经网络会产生不同的挖掘效果，这可能和选取特征的类型有关，而在这方面的研究依然很少。由此来看，深度学习在一定程度上能够提升“浅层”学习算法的效果，依靠深度学习模型的强大能力进行漏洞挖掘或许成为未来的主要漏洞挖掘方式。

3.1.6 跨项目漏洞挖掘

目前，大多数研究工作都集中在项目内漏洞挖掘（WPVP, within-project vulnerability prediction）中，跨项目漏洞挖掘（CPVP, cross-project vulnerability prediction）研究较少，同时研究的模型性能不够高。跨项目漏洞挖掘需要实现在一个项目上构造的漏洞挖掘模型用于挖掘另一个项目上的漏洞。在实际软件开发场景中，需要进行漏洞挖掘的项目可能是新启动的项目或这类项目的训练数据较为稀缺，这就需要对其进行跨项目漏洞挖掘。然而由于不同项目采用的开发流程、应用的领域、采用的编程语言、开发人员的经验等不同构成了跨项目漏洞挖掘的最大障碍。文献[32-33,59,61]仅仅粗浅地使用某个项目的软件度量、语义特征进行另一个项目的漏洞挖掘，并没有考虑编程语言、项目应用的差异性。

随着机器学习的发展，迁移学习被提出，用于将已经学到的模型参数通过某种方式来分享给新模型，从而加快并优化模型的学习效率，而不用像大多数神经网络那样从零学习，这为跨项目漏洞挖掘提供了较好的应用基础。Ma 等[88]对这方面进行了研究，提出了迁移朴素贝叶斯（TNB, transfer naive Bayes）模型，来对不同数据集进行缺陷检测训练，他们的结果证实了迁移学习能够取得不错的效果。文献[89]首次将迁移学习应用于漏洞挖掘，他们利用序列化的 AST 表征代码信息，采用双向 LSTM 实现跨项目的漏洞挖掘，并取得了不错的检测效果。文献[61]发现利用降维技术在跨项目漏洞挖掘中能够明显增强基于软件度量的模型效果，而在语义特征的模型中则没有太大变化。我们推测，在软件度量方面的跨项目漏洞检测中，降维技术保留了项目间的公共特性，提高了软件度量的跨项目

检测性能；而在语义特征方面，采用 Unigram 以及统计词频技术，这些语义模型本身就统计了项目中的一些公共属性，因此无法提高性能。

除此之外，跨项目漏洞挖掘局限于同种编程语言的不同项目之间进行漏洞挖掘，并没有实现跨语言漏洞挖掘，本文认为，跨语言漏洞挖掘需要对不同的编程语言进行映射，比如对函数定义、变量申请等代码语句进行转换，由此来实现跨语言漏洞挖掘。其次，还要考虑到不同的项目应用，比如一般项目中采用的加密算法和银行相关项目采用的加密算法在等级上的差异也会影响跨项目漏洞挖掘。综上，跨项目漏洞挖掘的研究目前处于初期阶段，对不同的语言、不同的应用环境，跨项目漏洞挖掘要具备不同的研究性能，这方面研究依然比较少。

3.2　自动化漏洞利用

实现漏洞利用的自动化生成是一个复杂的过程，首先要定位漏洞的位置，利用符号执行技术快速找到输入漏洞可利用点的路径，其次通过动态监控程序运行过程，能够获取程序实际运行时的栈布局信息，最后利用上述信息生成漏洞利用并进行验证。文献[90-98]研究了自动化漏洞利用生成（AEG, automatic exploit generation）。这些研究均取得一定的成果，能够自动化生成漏洞利用，然而这些技术能够处理的漏洞种类有限，同时它们并没有使用人工智能技术。You 等[99]提出了 SemFuzz，该框架首次利用 NLP 技术从 CVE 和 Git 日志中提取相关语义信息比如关键函数、变量来指导 PoC 的自动生成，同时扩大了处理漏洞的种类。他们的研究表明，将 NLP 应用到漏洞利用的自动化生成是可行的。除此之外，文献[100-102]采用机器学习算法从代码特征等方面对漏洞的可利用性进行了预测。

将人工智能技术引入漏洞利用的自动化生成领域，首先提取安全漏洞报告中包含与漏洞相关的软件名、版本号、涉及的函数以及漏洞类型等信息。这些信息能够初步安装软件并对漏洞进行粗略定位。其次利用人工智能技术与程序技术相结合能够加快分析过程。事实上，自动化漏洞利用生成依然存在漏洞信息不全面、信息利用率低、漏洞利用生成成功率低等问题。各大安全网站及安全论坛提供了相关的配置、软件依赖项等信息，这些信息也可以辅助漏洞利用的自动化生成。本文认为，利用 NLP 技术综合处理与漏洞相关的信息源，从而实现漏洞利用的自动化生成，这将成为漏洞利用自动化生成的新方法。通过对漏洞利用的自动化生成进行研究，对促进漏洞挖掘分析等均具有重大意义。

3.3　自动化漏洞评估

对安全漏洞进行评估能够帮助人们建立衡量漏洞严重程度的标准、确定漏洞修补的优先级。CVSS 将漏洞的严重程度分为 3 种等级，这为实现漏洞评估的自动化提供了基础。同时 CVE 漏洞库也能够为机器学习应用漏洞评估提供更多的特征选择，比如数据集、漏洞关键字、漏洞描述等信息。

Yamamoto 等[103]提出了将自然语言处理和机器学习算法结合的方法来自动化评估 CVE 文档的 CVSS 基准度量。而 Spanos 等[104]则通过对漏洞报告中关于漏洞的自然语言描述，从访问向量、访问复杂性、身份验证、识别性影响、完整性影响和可用性影响这 6 个方面进行文本挖掘，并以此来构建机器学习模型，实验结果均能够以 78%以上的准确度对漏洞进行评估。Han 等[105]将深度学习引入漏洞评估中，并提取 CVE 漏洞库中漏洞描述的相关文本特征来预测软件漏洞的严重程度。

研究人员从漏洞报告中提炼有效信息进行漏洞评估，取得了一定的效果，然而这些研究仅仅关注漏洞报告的信息，并没有考虑各大安全网站统计的信息。事实上，各大安全网站提供了有关安全漏洞的统计信息，比如 SecurityFocus 详细地统计了受影响的软件及其版本号。本文认为，除了上述有关漏洞报告的因素外，还要考虑到软件的应用范围，软件的应用范围越广，所产生的影响越大。其次，需要考虑软件的复杂度，越是复杂的软件越难以维护，漏洞持续的周期越长，造成的危害越大。因此 CVSS 提供了一种用于安全漏洞威胁严重等级评估的、公开免费的风险评估系统，然而其中的指标往往很复杂，难以快速实现漏洞评估。将自然语言处理与机器学习相结合对漏洞报告以及其他的漏洞信息源进行处理能够快速实现对安全漏洞的智能评估。

3.4　自动化漏洞修复

安全漏洞的修复是减少因安全漏洞暴露引起财产损失的最佳方法。从近几年的安全攻击时间来看，由于厂商更新速度赶不上漏洞 PoC 的传播速度，未能及时对漏洞进行修复，因此对计算机用户造成了较大的损失。实现漏洞的自动化修复，有助于快速弥补漏洞缺陷，减少用户财产损失，对促进计算机生态安全有重大作用。

Zhang 等[106]对整数溢出到缓冲区溢出（IO2BO,

integer overflow to buffer overflow）漏洞进行修补。Le 等[107]提出了 GenProg，使用遗传编程算法来生成新的程序变种，选择能够通过所有被考虑的测试用例的程序变种作为修补方案。White 等[108]对 GenProg 进行了扩展，提出了 DeepRepair，采用深度学习模型进行程序修补。Zhang 等[109]提出了一种自动补片技术 AppSealer，修复 Android 平台组件劫持漏洞。

将机器学习应用于漏洞自动化修补依然存在较大的困难。首先，漏洞准确定位是一个急需解决的难点，文献[23,31]研究了复杂性度量指标和其他指标，如过程度量能否进行漏洞定位。其次，漏洞的修补方式更复杂，既要确定触发漏洞的类型，又要对程序进行相应的修改，还要保证修补后的程序能够正常运行并且保证不会产生其他漏洞。最后，由于目前程序分析技术无法对软件进行全面的分析，还需要人工进行漏洞修复。Ben 等[110]的研究确定了 8 类共 65 个影响漏洞修复时间的因素。他们的工作能够改进漏洞修复过程，更合理地配置漏洞修复资源，提高漏洞修复效率。部分漏洞如整数溢出漏洞以及格式化字符串漏洞能够采用故障修补方式进行修补。由此可见，人工进行漏洞修补方式依然是漏洞修补的主流方式，实现漏洞修补的自动化还需要研究人员投入大量的研究。

3.5 小结

安全漏洞的自动化研究始终是网络空间安全研究的重点，将人工智能技术引入安全漏洞研究，有利于促进安全漏洞研究的自动化发展。本节通过将人工智能技术应用到安全漏洞研究过程，分析了自动化漏洞挖掘、自动化漏洞利用、自动化漏洞评估、自动化漏洞修复等一系列新的研究成果，同时指出了现有的研究存在的问题，并给出了一些建议以及解决方法。

4 未来研究展望

将人工智能技术应用于安全漏洞研究一直是网络空间安全研究的重要方向之一，这对计算机系统安全和网络空间安全有着非常重要的意义。然而，由于漏洞种类繁多，漏洞产生原理、触发条件彼此不一致，因此对安全漏洞无法进行统一有效的挖掘。由此可见，实现漏洞自动化挖掘等研究过程还有很长的路要走。将人工智能技术应用于安全漏洞研究的一些问题以及可能的解决方法如表 3 所示。

表 3 人工智能技术应用于安全漏洞研究面临的问题与机遇

挑战	机遇
软件度量	新的代码属性
语义特征发展	新的语义模型
语义信息爆炸	降维处理
漏洞定位	细粒度的程序表征方式
高误报或者高漏报	将 ML 与静态、动态分析相结合
机器学习算法	深度学习算法
跨项目漏洞挖掘	迁移学习
漏洞数据集	建立一个公开的、可以作为基准的数据集
自动化漏洞利用生成	利用 NLP 技术辅助漏洞、利用自动化生成
漏洞智能评估	结合 NLP 技术与 ML 技术实现漏洞智能评估

1) 特征选择

在漏洞挖掘方面，本文根据选择特征的不同将其分为基于软件度量的漏洞挖掘模型以及基于语法语义特征的漏洞挖掘模型。对于机器学习算法而言，选取的特征越能代表数据集的特性，那么构造的模型越能拥有更好的性能。因此，基于软件度量的漏洞挖掘模型需要开发新的代码属性特征来更好地进行漏洞挖掘。代码属性作为一种有前景的特征需要研究人员继续进行深入研究，同时语义特征可以考虑采用新型的语义模型表征程序进行漏洞挖掘，或利用 NLP 技术、深度学习等技术提取程序的有效信息进行漏洞挖掘。同时，对语义模型中的特征爆炸问题，可以采用降维方法对数据处理来提高模型的性能。无论是软件度量还是语义特征，它们均属于漏洞挖掘的不同方面，能否将二者进行融合进行漏洞挖掘也是一个难点。而在漏洞评估方面，目前尚未找到有效的特征进行漏洞评估，因此还需要开发有效的软件特征进行相应的预测。

2) 深度学习模型

深度学习模型是近年来提出的新的机器学习模型，将深度学习应用于漏洞研究目前还处于起步阶段，借助深度学习模型强大的性能，将深度学习模型应用于漏洞挖掘、漏洞利用、漏洞评估与漏洞修复等均属于难点问题。而深度模型应用于漏洞挖掘面临的首要问题是如何将程序表征转化为适合深度模型的向量表达；其次在于检测粒度，细粒度的漏洞挖掘模型能够识别漏洞位置，这扩展了安全漏洞的研究；最后在于深度学习算法，目前深度学习算法也有很多种，不同的算法对不同的特征会产生不同的结果，如何选择特征与深度学习模型也是

一个研究难点，而目前这些方面研究很少。

3) 跨项目检测

跨项目漏洞挖掘是对不同项目的漏洞进行检测，由于编程语言、应用领域等差异，导致跨项目漏洞挖掘的难度很大。目前，实现跨项目漏洞挖掘的研究较少，其模型性能较低。迁移学习作为解决跨项目漏洞挖掘的有效手段依然存在一些研究难点，比如如何利用迁移学习来实现跨语言漏洞挖掘以及不同领域的跨项目漏洞挖掘等。

4) 数据集

将机器学习应用于漏洞挖掘，首要考虑的是数据集。在本文调研过程中发现，现有的研究数据集可以分为组件级数据集、函数级数据集以及代码级数据集。函数级数据集以及组件级数据集常用于基于语义语法特征的漏洞挖掘模型，而代码级数据集则倾向于基于软件度量的漏洞挖掘模型。事实上，不同的数据集对漏洞挖掘模型的性能也会产生不同的影响，细粒度的数据集更有益于漏洞位置的识别，同时也加大了计算处理的难度。目前并没有一个公开的可以作为基准的漏洞数据集。因此本文认为将机器学习引入漏洞挖掘，必须建立一个公开的、能够作为测试基准的数据集。

5) 高漏报或者高误报

常见的程序分析技术如静态分析、动态分析等技术在漏洞挖掘过程中起到了很大的作用，然而这些技术往往面临着高漏报率或高误报率等问题。同时，符号执行、污点分析等技术也存在着各自的缺点。机器学习通过对大量的样本进行训练能够从中抽象出相应的特征，并能够筛选出可能存在问题函数或者代码片段。因此，将机器学习同静态分析、动态分析等分析技术相结合进行漏洞挖掘来降低模型的误报率或漏报率，提高漏洞模型的准确率。同时，将机器学习同程序分析技术相结合为解决约束求解难、路径执行空间爆炸等问题提供了新的思路。将机器学习同静态分析技术、动态分析技术结合进行漏洞挖掘也是一个可探讨的研究方向。

5　结束语

随着人工智能技术的不断发展，利用人工智能技术对软件进行分析，实现软件的安全漏洞研究成为安全研究的重要方向之一。本文总结了人工智能技术应用于安全漏洞研究方面最新的应用，归纳了其存在的问题并进行了相应的探讨。本文认为，在未来将人工智能技术应用于漏洞的研究中，深度学习模型能够促进安全漏洞的研究，开发新的漏洞特征则能够提高现有漏洞挖掘模型的准确度，同时对漏洞定位、漏洞挖掘中的漏报误报等问题进一步分析，提高人工智能技术在漏洞研究方面起到的作用，这对促进智能化漏洞研究均有重大影响。

参考文献：

[1] 张玉清，宫亚峰，王宏，等.安全漏洞标识与描述规范[S]. GB/T28458-2012, 全国信息安全标准化技术委员会(SAC/TC 260). ZHANG Y Q,GONG Y F,WANG H, et al. Vulnerability identification and description specification[S]. GB/T28458-2012, National Information Security Standardization Technical Committee.

[2] WITTEN I H, FRANK E, HALL M A, et al. Data mining: practical machine learning tools and techniques[M]. Morgan Kaufmann, 2016.

[3] VAPNIK V N. An overview of statistical learning theory[J]. IEEE transactions on neural networks, 1999, 10(5): 988-999.

[4] NASRABADI N M. Pattern recognition and machine learning[J]. Journal of Electronic Imaging, 2007, 16(4): 049901.

[5] MITCHELL T M. Machine learning and data mining[J]. Communications of the ACM, 1999, 42(11): 30-36.

[6] LECUN Y, BENGIO Y, HINTON G. Deep learning[J]. Nature, 2015, 521(7553): 436.

[7] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks[J]. Communications of the ACM, 2012, 60(2): 2012-2025.

[8] TAIGMAN Y, YANG M, RANZATO M A, et al. Deepface: closing the gap to human-level performance in face verification[C]//The 29th IEEE Conference on Computer Vision and Pattern Recognition. 2014: 1701-1708.

[9] COLLOBERT R, WESTON J. A unified architecture for natural language processing: deep neural networks with multitask learning[C]//The 25th International Conference on Machine Learning. 2008: 160-167.

[10] HUANG W Y, STOKES J W. MtNet: a multi-task neural network for dynamic malware classification[C]//The 5th 25th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. 2016:399-418.

[11] DEBAR H, BECKER M, SIBONI D. A neural network component for an intrusion detection system[C]//The 23rd Computer Society Symp on Research in Security and Privacy. 1992: 240-250.

[12] CEARA D, POTET M L, ENSIMAG G I N P, et al. Detecting software vulnerabilities-static taint analysis[J]. Polytechnic University of Bucharest, 2009.

[13] KING J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394.

[14] GAO D, REITER M K, SONG D. Binhunt: automatically finding semantic differences in binary programs[C]//International Conference on Information and Communications Security. 2008: 238-255.

[15] DUKES L S, YUAN X, AKOWUAH F. A case study on web application security testing with tools and manual testing[C]//2013 Proceedings of IEEE. 2013: 1-6.

[16] SUTTON M, GREENE A, AMINI P. Fuzzing: brute force vulnerability discovery[M]. Pearson Education, 2007.

[17] NEWSOME J, SONG D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[J]. 2005.

[18] XIE T, TILLMANN N, DE H J, et al. Fitness-guided path exploration in dynamic symbolic execution[C]//IEEE/IFIP International Conference on Dependable Systems & Networks. 2009: 359-368.

[19] SURHONE L M, TENNOE M T, HENSSONOW S F, et al. Common vulnerabilities and exposures[M]. Betascript Publishing, 2010.

[20] MELL P, SCARFONE K, ROMANOSKY S. Common vulnerability scoring system[J]. IEEE Security & Privacy, 2006, 4(6).

[21] MCCABE T J. A complexity measure[J]. IEEE Transactions on software Engineering, 1976 (4): 308-320.

[22] HALSTEAD M H. Elements of software science (operating and programming systems series)[M]. Elsevier Science Inc, 1977.

[23] ZIMMERMANN T, NAGAPPAN N, WILLIAMS L. Searching for a needle in a haystack: predicting security vulnerabilities for windows vista[C]// 2010 Third International Conference on Software Testing, Verification and Validation (ICST). 2010: 421-428.

[24] SHIN Y, WILLIAMS L. Can traditional fault prediction models be used for vulnerability prediction?[J]. Empirical Software Engineering, 2013, 18(1): 25-59.

[25] SHIN Y, WILLIAMS L. An empirical model to predict security vulnerabilities using code complexity metrics[C]//The Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. 2008: 315-317.

[26] SHIN Y, WILLIAMS L. Is complexity really the enemy of software security?[C]//The 4th ACM Workshop on Quality of Protection. 2008: 47-50.

[27] SHIN Y, WILLIAMS L. An initial study on the use of execution complexity metrics as indicators of software vulnerabilities[C]//The 7th International Workshop on Software Engineering for Secure Systems. 2011: 1-7.

[28] DOYLE M, WALDEN J. An empirical study of the evolution of PHP web application security[C]//2011 Third International Workshop on Security Measurements and Metrics (Metrisec). 2011: 11-20.

[29] CHOWDHURY I, ZULKERNINE M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities[J]. Journal of Systems Architecture, 2011, 57(3): 294-313.

[30] CHOWDHURY I, ZULKERNINE M. Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?[C]//The 2010 ACM Symposium on Applied Computing. 2010: 1963-1969.

[31] SHIN Y, MENEELY A, WILLIAMS L, et al. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities[J]. IEEE Transactions on Software Engineering, 2011, 37(6): 772-787.

[32] MENEELY A, WILLIAMS L. Strengthening the empirical analysis of the relationship between Linus' Law and software security[C]//The 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. 2010: 9.

[33] MOSHTARI S, SAMI A, AZIMI M. Using complexity metrics to improve software security[J]. Computer Fraud & Security, 2013, 2013(5): 8-17.

[34] ALVES H, FONSECA B, ANTUNES N. Software metrics and security vulnerabilities: dataset and exploratory study[C]//Dependable Computing Conference (EDCC). 2016: 37-44.

[35] MORRISON P, HERZIG K, MURPHY B, et al. Challenges with applying vulnerability prediction models[C]//The 2015 Symposium and Bootcamp on the Science of Security. 2015: 4.

[36] SCANDARIATO R, WALDEN J. Predicting vulnerable classes in an Android application[C]//The 4th International Workshop on Security Measurements and Metrics. 2012: 11-16.

[37] MENEELY A, SRINIVASAN H, MUSA A, et al. When a patch goes bad: Exploring the properties of vulnerability-contributing commits[C]//2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2013: 65-74.

[38] PERL H, DECHAND S, SMITH M, et al. Vccfinder: finding potential vulnerabilities in open-source projects to assist code audits[C]//The 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015: 426-437.

[39] YANG L, LI X, YU Y. VulDigger: a just-in-time and cost-aware tool for digging vulnerability-contributing changes[C]//2017 IEEE Global Communications Conference. 2017: 1-7.

[40] GEGICK M, WILLIAMS L, OSBORNE J, et al. Prioritizing software security fortification throughcode-level metrics[C]//The 4th ACM workshop on Quality of Protection. 2008: 31-38.

[41] NGUYEN V H, TRAN L M S. Predicting vulnerable software components with dependency graphs[C]//The 6th International Workshop on Security Measurements and Metrics. 2010: 3.

[42] YAN H, SUI Y, CHEN S, et al. Machine-learning-guided typestate analysis for static use-after-free detection[C]//The 33rd Annual Computer Security Applications Conference. 2017: 42-54.

[43] GUPTA M K, GOVIL M C, SINGH G. Predicting cross-site scripting (XSS) security vulnerabilities in web applications[C]//2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE). 2015: 162-167.

[44] ZHANG S, CARAGEA D, OU X. An empirical study on using the national vulnerability database to predict software vulnerabilities[C]//International Conference on Database and Expert Systems Applications. 2011: 217-231.

[45] SHAR L K, TAN H B K. Predicting common web application vulnerabilities from input validation and sanitization code patterns[C]//The 27th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2012: 310-313.

[46] SHAR L K, TAN H B K. Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns[J]. Information and Software Technology, 2013, 55(10): 1767-1780.

[47] SHAR L K, TAN H B K, BRIAND L C. Mining SQL injection and cross site scripting vulnerabilities using hybrid program analy-

sis[C]//The 2013 International Conference on Software Engineering. 2013: 642-651.

[48] SHAR L K, BRIAND L C, TAN H B K. Web application vulnerability prediction using hybrid program analysis and machine learning[J]. IEEE Transactions on Dependable and Secure Computing, 2015, 12(6): 688-707.

[49] PADMANABHUNI B M, TAN H B K. buffer overflow vulnerability prediction from x86 executables using static analysis and machine learning[C]//Computer Software and Applications Conference (COMPSAC). 2015: 450-459.

[50] PADMANABHUNI B M, TAN H B K. Predicting buffer overflow vulnerabilities through mining light-weight static code attributes[C]// 2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). 2014: 317-322.

[51] PADMANABHUNI B M, TAN H B K. Auditing buffer overflow vulnerabilities using hybrid static-dynamic analysis[J]. IET Software, 2016, 10(2): 54-61.

[52] MENG Q, ZHANG B, FENG C, et al. Detecting buffer boundary violations based on SVM[C]//2016 3rd International Conference on Information Science and Control Engineering (ICISCE). 2016: 313-316.

[53] MENG Q, WEN S, FENG C, et al. Predicting integer overflow through static integer operation attributes[C]//International Conference on Computer Science and Network Technology. 2017:177-181.

[54] WANG D, LIN M, ZHANG H, et al. Detect related bugs from source code using bug information[C]//Computer Software and Applications Conference (COMPSAC). 2010: 228-237.

[55] HOVSEPYAN A, SCANDARIATO R, JOOSEN W, et al. Software vulnerability prediction using text analysis techniques[C]//The 4th International Workshop on Security Measurements and Metrics. 2012: 7-10.

[56] SCANDARIATO R, WALDEN J, HOVSEPYAN A, et al. Predicting vulnerable software components via text mining[J]. IEEE Transactions on Software Engineering, 2014, 40(10): 993-1006.

[57] PANG Y, XUE X, NAMIN A S. Early identification of vulnerable software components via ensemble learning[C]//2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA). 2016: 476-481.

[58] DAM H K, TRAN T, PHAM T, et al. Automatic feature learning for vulnerability prediction[J]. arXiv preprint, arXiv:1708.02368, 2017.

[59] PANG Y, XUE X, NAMIN A S. Predicting vulnerable software components through n-gram analysis and statistical feature selection[C]// 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). 2015: 543-548.

[60] PANG Y, XUE X, WANG H. Predicting vulnerable software components through deep neural network[C]//The 2017 International Conference on Deep Learning Technologies. 2017: 6-10.

[61] STUCKMAN J, WALDEN J, SCANDARIATO R. The effect of dimensionality reduction on software vulnerability prediction models[J]. IEEE Transactions on Reliability, 2017, 66(1): 17-37.

[62] WIJAYASEKARA D, MANIC M, WRIGHT J L, et al. Mining bug databases for unidentified software vulnerabilities[C]// 2012 5th International Conference on Human System Interactions (HSI). 2012: 89-96.

[63] WIJAYASEKARA D, MANIC M, MCQUEEN M. Information gain based dimensionality selection for classifying text documents[C]// 2013 IEEE Congress on Evolutionary Computation (CEC). 2013: 440-445.

[64] WIJAYASEKARA D, MANIC M, MCQUEEN M. Vulnerability identification and classification via text mining bug databases[C]//Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE. 2014: 3612-3618.

[65] YAMAGUCHI F, LOTTMANN M, RIECK K. Generalized vulnerability extrapolation using abstract syntax trees[C]//The 28th Annual Computer Security Applications Conference. 2012: 359-368.

[66] YAMAGUCHI F, WRESSNEGGER C, GASCON H, et al. Chucky: exposing missing checks in source code for vulnerability discovery[C]//The 2013 ACM SIGSAC Conference on Computer & Communications Security. 2013: 499-510.

[67] YAMAGUCHI F, MAIER A, GASCON H, et al. Automatic inference of search patterns for taint-style vulnerabilities[C]// 2015 IEEE Symposium on Security and Privacy (SP). 2015: 797-812.

[68] MENG Q, WEN S, ZHANG B, et al. Automatically discover vulnerability through similar functions[C]//Progress in Electromagnetic Research Symposium (PIERS). 2016: 3657-3661.

[69] MEDEIROS I, NEVES N, CORREIA M. Detecting and removing web application vulnerabilities with static analysis and data mining[J]. IEEE Transactions on Reliability, 2016, 65(1): 54-69.

[70] MENG Q, SHAMENG W, CHAO F, et al. Predicting buffer overflow using semi-supervised learning[C]// International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2016: 1959-1963.

[71] ALOHALY M, TAKABI H. When do changes induce software vulnerabilities?[C]//2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC). 2017: 59-66.

[72] NEUHAUS S, ZIMMERMANN T, HOLLER C, et al. Predicting vulnerable software components[C]//The 14th ACM Conference on Computer and Communications Security. 2007: 529-540.

[73] YAMAGUCHI F, LINDNER F, RIECK K. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning[C]//The 5th USENIX Conference on Offensive Technologies. 2011: 13.

[74] WALDEN J, STUCKMAN J, SCANDARIATO R. Predicting vulnerable components: software metrics vs text mining[C]//2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE). 2014: 23-33.

[75] TANG Y, ZHAO F, YANG Y, et al. Predicting vulnerable components via text mining or software metrics? an effort-aware perspective[C]// 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS). 2015: 27-36.

[76] ZHANG Y, LO D, XIA X, et al. Combining software metrics and text features for vulnerable file prediction[C]//2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS). 2015: 40-49.

[77] MENG Q, ZHANG B, FENG C, et al. Detecting buffer boundary violations based on SVM[C]// 2016 3rd International Conference on Information Science and Control Engineering (ICISCE). 2016: 313-316.

[78] MEDEIROS I, NEVES N F, CORREIA M. Automatic detection and

correction of web application vulnerabilities using data mining to predict false positives[C]//The 23rd International Conference on World Wide Web. 2014: 63-74.

[79] HEO K, OH H, YI K. Machine-learning-guided selectively unsound static analysis[C]//The 39th International Conference on Software Engineering. 2017: 519-529.

[80] GRIECO G, GRINBLAT G L, UZAL L, et al. Toward large-scale vulnerability discovery using machine learning[C]//The Sixth ACM Conference on Data and Application Security and Privacy. 2016: 85-96.

[81] GODEFROID P, PELEG H, SINGH R. Learn&fuzz: machine learning for input fuzzing[C]//The 32nd IEEE/ACM International Conference on Automated Software Engineering. 2017: 50-59.

[82] LESSMANN S, BAESENS B, MUES C, et al. Benchmarking classification models for software defect prediction: a proposed framework and novel findings[J]. IEEE Transactions on Software Engineering, 2008, 34(4): 485-496.

[83] GHOTRA B, MCINTOSH S, HASSAN A E. Revisiting the impact of classification techniques on the performance of defect prediction models[C]//The 37th International Conference on Software Engineering. 2015: 789-800.

[84] TANTITHAMTHAVORN C, MCINTOSH S, HASSAN A E, et al. Automated parameter optimization of classification techniques for defect prediction models[C]//2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). 2016: 321-332.

[85] MOU L, LI G, LIU Y, et al. Building program vector representations for deep learning[J]. arXiv preprint, arXiv:1409.3358, 2014.

[86] LI Z, ZOU D, XU S, et al. VulDeePecker: a deep learning-based system for vulnerability detection[J]. arXiv preprint, arXiv:1801.01681, 2018.

[87] WU F, WANG J, LIU J, et al. Vulnerability detection with deep learning[C]//2017 3rd IEEE International Conference on Computer and Communications (ICCC). 2017: 1298-1302.

[88] MA Y, LUO G, ZENG X, et al. Transfer learning for cross-company software defect prediction[J]. Information and Software Technology, 2012, 54(3): 248-256.

[89] LIN G, ZHANG J, LUO W, et al. Cross-project transfer representation learning for vulnerable function discovery[J]. IEEE Transactions on Industrial Informatics, 2018.

[90] BRUMLEY D, POOSANKAM P, SONG D, et al. Automatic patch-based exploit generation is possible: techniques and implications[C]//IEEE Symposium on Security and Privacy. 2008: 143-157.

[91] CHA S K, AVGERINOS T, REBERT A, et al. Unleashing mayhem on binary code[C]//2012 IEEE Symposium on Security and Privacy (SP). 2012: 380-394.

[92] WANG M, SU P, LI Q, et al. Automatic polymorphic exploit generation for software vulnerabilities[C]//International Conference on Security and Privacy in Communication Systems. 2013: 216-233.

[93] HU H, CHUA Z L, ADRIAN S, et al. Automatic generation of data-oriented exploits[C]//USENIX Security Symposium. 2015: 177-192.

[94] BAO T, WANG R, SHOSHITAISHVILI Y, et al. Your exploit is mine: automatic shellcode transplant for remote exploits[C]//2017 IEEE Symposium on Security and Privacy (SP). 2017: 824-839.

[95] ALHUZALI A, ESHETE B, GJOMEMO R, et al. Chainsaw: chained automated workflow-based exploit generation[C]// ACM Sigsac Conference on Computer and Communications Security. 2016:641-652.

[96] HUANG S K, LU H L, LEONG W M, et al. CRAXweb: automatic web application testing and attack generation[C]//IEEE, International Conference on Software Security and Reliability. 2013:208-217.

[97] FELMETSGER V, CAVEDON L, KRUEGEl C, et al. Toward automated detection of logic vulnerabilities in Web applications[C]// Usenix Security Symposium. 2010:143-160.

[98] LUO L, ZENG Q, CAO C, et al. System service call-oriented symbolic execution of android framework with applications to vulnerability discovery and exploit generation[C]//The 15th Annual International Conference on Mobile Systems, Applications, and Services. 2017: 225-238.

[99] YOU W, ZONG P, CHEN K, et al. SemFuzz: semantics-based automatic generation of proof-of-concept exploits[C]//The 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2139-2154.

[100] YOUNIS A, MALAIYA Y, ANDERSON C, et al. To fear or not to fear that is the question: code characteristics of a vulnerable functionwith an existing exploit[C]//The Sixth ACM Conference on Data and Application Security and Privacy. 2016: 97-104.

[101] BOZORGI M, SAUL L K, SAVAGE S, et al. Beyond heuristics: learning to classify vulnerabilities and predict exploits[C]//The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2010: 105-114..

[102] ALLODI L, MASSACCI F. A preliminary analysis of vulnerability scores for attacks in wild: the ekits and sym datasets[C]//The 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security. 2012: 17-24.

[103] YAMAMOTO Y, MIYAMOTO D, NAKAYAMA M. Text-mining approach for estimating vulnerability score[C]// International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. 2017:67-73.

[104] SPANOS G, ANGELIS L, TOLOUDIS D. Assessment of vulnerability severity using text mining[C]// Pan-Hellenic Conference on Informatics. 2017:1-6.

[105] HAN Z, LI X, XING Z, et al. Learning to predict severity of software vulnerability using only vulnerability description[C]// 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2017: 125-136.

[106] ZHANG C, WANG T, WEI T, et al. IntPatch: automatically fix integer-overflow-to-buffer-overflow vulnerability at compile-time[C]// European Symposium on Research in Computer Security. 2010: 71-86.

[107] LE G C, NGUYEN T V, FORREST S, et al. Genprog: a generic method for automatic software repair[J]. IEEE Transactions On Software Engineering, 2012, 38(1): 54-72.

[108] WHITE M, TUFANO M, MARTINEZ M, et al. Sorting and transforming program repair ingredients via deep learning code similarities[J]. arXiv preprint, arXiv:1707.04742, 2017.

[109]ZHANG M, YIN H. AppSealer: automatic generation of vulnerability-specific patches for preventing component hijacking attacks in android applications[C]//NDSS. 2014.

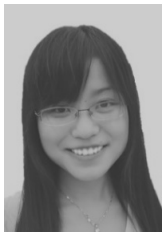
[110]BEN O L, CHEHRAZI G, BODDEN E, et al. Factors impacting the effort required to fix security vulnerabilities[C]//International Information Security Conference. 2015: 102-119.

[作者简介]

孙鸿宇（1993−），男，陕西渭南人，西安电子科技大学博士生，主要研究方向为信息安全与机器学习。

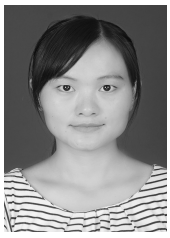
何远（1977−），男，云南大理人，中国科学院大学博士生，主要研究方向为计算机信息安全与漏洞挖掘。

王基策（1992−），男，河南襄城人，中国科学院大学博士生，主要研究方向为移动安全、软件安全等。

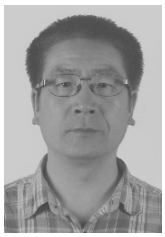


董颖（1991−），女，陕西渭南人，中国科学院大学博士生，主要研究方向为网络安全和机器学习。

朱立鹏（1994−），男，河北秦皇岛人，西安电子科技大学硕士生，主要研究方向为物联网安全和漏洞挖掘。



王鹤（1987−），女，河南安阳人，博士，西安电子科技大学讲师，主要研究方向为量子密码协议。



张玉清（1966−），男，陕西宝鸡人，博士，中国科学院大学教授、博士生导师，主要研究方向为网路与信息系统安全。