软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

自然语言数据驱动的智能化软件安全评估方法
(数据驱动的软件智能化开发方法与技术专刊)*

张一帆[1,2], 汤恩义[1,3*], 苏琰梓[1,3], 杨开懋[1,3], 匡宏宇[1,3], 陈 鑫[1,2]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210023)

[2](南京大学 计算机科学与技术系,江苏 南京 210023)

[3](南京大学 软件学院,江苏 南京 210093)

通讯作者: 汤恩义, E-mail: eytang@nju.edu.cn

摘 要: 软件安全性是衡量软件是否能够抵御恶意攻击的重要性质.在当前互联网环境下,黑客攻击无处不在,因而估计软件中可能含有的漏洞数量与类型,即对软件进行安全评估,变得十分必要.在实际中用户不仅需要对未发布、或者最新发布的软件实施安全性评估,对已发布软件也会有一定的安全评估需求,例如当用户需要从市场上互为竞争的多款软件中作出选择,就会希望能花费较低成本、较为客观地对这些软件进行第三方的评估与比较.本文提出了一种由自然语言数据驱动的智能化软件安全评估方法来满足这一要求,该方法基于待评估软件现有用户的使用经验信息来评估软件的安全性,它首先自适应地爬取用户在软件使用过程中对软件的自然语言评价数据,并利用深度学习方法与机器学习评估模型的双重训练来获得软件的安全性评估指标.由于本文的自适应爬虫能够在反馈中调整特征词,并结合搜索引擎来获得异构数据,因而可通过采集广泛的自然语言数据来进行安全评估.另外,使用一对多的机器翻译训练能有效解决将自然语言数据转换为语义编码的问题,使得用于安全评估的机器学习模型可以建立在自然语言的语义特征基础上.我们进一步在国际通用漏洞披露数据库(CVE)和美国国家漏洞数据库(NVD)上对本文方法进行了实验,结果表明,本文方法在评估软件漏洞数量,漏洞类型,以及漏洞严重程度等指标上十分有效.

关键词: 软件安全评估;自然语言处理;机器学习;网络爬虫

中图法分类号: TP311

中文引用格式: 张一帆,汤恩义,苏琰梓,杨开懋,匡宏宇,陈鑫.自然语言数据驱动的智能化软件安全评估方法.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Zhang YF,Tang EY,Su YZ,Yang KM,Kuang HY,Chen X. A Natural Language Data Driven Approach for Software Intelligent Safety Evaluation. Ruan Jian Xue Bao/Journal of Software, 2016 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

A Natural Language Data Driven Approach for Software Intelligent Safety Evaluation

ZHANG Yi-Fan[1,2], TANG En-Yi[1,3*], SU Yan-Zi[1,3], YANG Kai-Mao[1,3], KUANG Hong-Yu[1,3], CHEN Xin[1,2]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

[3](Software Institute, Nanjing University, Nanjing 210093, China)

+ Corresponding author:TANG En-Yi. E-mail:eytang@nju.edu.cn

* 基金项目: 国家自然科学基金(00000000, 00000000); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT00000000)

Foundation item: National Natural Science Foundation of China (00000000, 00000000); State Key Laboratory for Novel Software Technology (Nanjing University)开放课题 (KFKT00000000)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

Abstract: Software safety is a key property that determines whether software is vulnerable to malicious attacks. Nowadays, internet attacks are ubiquitous, thus it is important to evaluate the number and category of defects in the software. Users need not only evaluate the safety of software that is released recently, or even is not released yet, but also evaluate the software that is already published for a while. For example, when users want to evaluate the safety of several competitive software systems before they decide their purchase, they need a low cost, objective evaluation approach. In this paper, we propose a natural language data driven approach for evaluating the safety of software that is released already. Our approach crawls natural language data adaptively, and applies a dual training to evaluate the software safety. As our self-adaptive web crawler adjusts feature words from the feedback and acquires heterogeneous data from search engines, our software safety evaluation utilizes extensive data sources automatically. Furthermore, by customizing a machine translation model, it is quite efficient to convert natural language to its semantic encoding. Hence, we build a machine learning model for intelligently evaluating software safety based on semantic characteristics of natural language. We conduct some experiments on the Common Vulnerabilities and Exposures (CVE) and the National Vulnerability Database (NVD). The results show that our approach is able to make safety evaluations precisely on the amount, impact and category of defects in software.

Key words: software safety evaluation; natural language processing; machine learning; web crawler

1 引言

软件安全性是指软件在遭受恶意攻击的情况下仍能保持其功能获得正确运行的性质.许多软件在设计与开发过程中都未能充分考虑其安全性,从而导致其或多或少存在一些可以被利用的安全漏洞.攻击者可以在未授权的情况下通过这些安全漏洞访问或者破坏系统,使用户遭受损失.而在当前互联网与移动计算环境下,黑客攻击日益频繁,软件的安全性威胁也在逐年增加.据不完全统计[1,2],2016 年被检测到的网络攻击数比 2011 年增长了 48 倍.不仅如此,黑客的攻击手段日趋多样,横跨了云计算、物联网以及移动计算等各个平台,攻击技术也在不断翻新,各种新类型的安全性问题不断涌现.因此,软件是否安全是使用与部署软件所必须考虑的重要方面.

软件安全评估是针对软件安全性的度量,即评价软件中可能存在的漏洞,以及因可能遭受的安全攻击而造成损失的风险.基于这一度量,用户可以预测软件中存在的潜在安全威胁,并针对这些威胁做好必要防范措施,以避免或减少因为软件安全问题而造成的损失.目前已有的安全性评估方法大多针对未发布、或者最新发布的软件,其评估过程一方面依赖于静态程序分析与动态软件测试等基于代码的指标提取与确认技术,另一方面依赖于人工对评估指标进行进一步的总结与分析,因此评估成本较高.用户对已发布软件也会有一定的安全性评估要求,例如当用户希望对市场上互为竞争的多款软件或服务进行选择,采用其中某一款软件来满足自己的功能需求时,需要较为客观地了解这几款软件的安全性状况.此时,作为软件的新用户,希望能够花费较低的成本对这些软件进行第三方评估与比较,而不是由软件发布方直接提供自己软件的安全性评估数据.

已有的安全性评估方法往往并不适用于这一评估场景.首先,用户往往获取不到这些待评估软件的源代码,甚至取得的二进制码也是经过代码混淆的,很难部署基于代码的分析技术来提取与确认所需的评估指标.其次,随着软件规模的增长，软件功能逻辑日益复杂,程序分析与软件测试的运行成本也随之提高.例如:基于静态分析的软件安全评估常依赖于上下文敏感、控制流敏感或者路径敏感的程序分析技术,而这些技术的复杂度往往随着软件规模的增加呈指数级增长.许多面向大规模软件的程序分析技术为了降低计算复杂度而约简了上下文、控制流以及路径信息,使得分析精度大为下降.再次,聘请安全专家来人工对每一款软件的安全性进行总结与分析十分昂贵,常常超出了用户的负担能力.因此,用户需要一种不依赖于代码分析的智能化软件安全评估方法来对已发布软件进行安全性评估.

近年来,深度学习相关的自然语言智能处理技术使计算机对于自然语言的处理能力上了一个新的台阶.基于这一现状,本文提出了一种由自然语言数据驱动的智能化软件安全评估方法,该方法基于待评估软件现有用户的使用经验来评估已发布软件的安全性.当待评估软件已经发布了一段时间,且有一定的用户数量以后,用户会自发地在互联网上交流该软件的使用心得,描述软件系统的特性,甚至对软件系统进行一些主观评价等等.事实上,用户对软件的使用心得客观上反映了待评估软件的安全特性,通过采用智能化技术对一定数量的用户描述信息进行统计分析,即可以获得较为准确的评估结果,本文正是基于这一原理来进行软件安全评估的.

本文自然语言数据驱动的智能化软件安全评估方法须克服以下难点:1.驱动安全评估的实际信息隐含在用户之间相互交流的自然语言数据之中,计算机须以智能化的方式来进行处理;2.支撑软件安全评估的自然语言数据分散在各个异构的交流平台和交流软件中,这导致了我们不能以简单的方式来直接获得这些自然语言数据.例如,有些系统的安全特性描述存在于用户正式的软件缺陷报告里,而另一些软件的描述信息可能隐含在用户的技术交流论坛中,这要求我们的安全评估方法必须能够智能化地突破这些异构平台限制来收集自然语言数据.

我们结合了网络爬虫技术与自然语言智能化处理技术来突破这些技术难点的限制.对于第一个难点,本文引入了一种基于深度学习的自然语言处理技术,该技术基于自然语言语料库与安全漏洞报告库的双重训练,从而能有效提取隐含在用户自然语言信息中的软件安全特性.这里我们借鉴了自然语言处理领域的深度学习模型,并在机器翻译模型的基础上,使用一对多(即一种源语言对多种目标语言)的翻译模式来训练循环神经网络,从而使该神经网络模型能获得较为准确的段落语义编码.并由该语义编码进一步训练针对安全评估的机器学习模型.对于第二个难点,我们构建了一个针对软件安全评估的网络爬虫.在爬虫与搜索引擎的交互阶段,我们通过反馈的方式构建自适应爬取特征词,通过结合搜索引擎与自适应爬取特征词,我们的爬虫会自动爬取不同类型的自然语言数据,从而克服了异构平台的限制,并解决了数据来源等问题.

我们进一步在国际通用漏洞披露数据库[1](Common Vulnerabilities and Exposures, CVE)和美国国家漏洞数据库[2](National Vulnerability Database, NVD)上对本文方法进行了实验评估.这些数据库中准确披露了一些已知软件的漏洞数量,漏洞类型,以及漏洞的严重程度.实验结果表明, 基于自然语言处理的方法可以有效驱动智能化软件安全评估的进行.相对于其它机器学习算法,随机森林方法在自然语言数据驱动的智能化软件安全评估中表现突出.例如:该算法对于各软件存在漏洞最多的安全漏洞类型,其预测准确率可以达到 82.53%.对于自适应爬虫来说,我们的实验发现采用 2 个特征词来结合搜索引擎可以达到最低的预测误差.而使用不同的搜索引擎在预测效果上差别不大,在平均误差上仅仅相差了 3.3 个百分点.

2 评估原理

近年来,基于深度学习的自然语言智能处理技术使计算机对于自然语言的处理能力上了一个新的台阶,使得机器自动翻译,自动问题解答,语义主题分类等传统的自然语言处理应用取得重要突破[3].深度学习的主要优势在于可以避免传统机器学习过程中需要手动设定数据特征的过程,例如传统机器学习中可能需要人工设定关键词来代表关键语义,人工在预处理阶段生成词频特征来加强机器学习的效果等等,这些人工设定的数据特征具有很强的经验性与主观性.而在深度学习算法下,通过大数据的训练,深度学习模型会自动获得更有表达力的特征,从而避免了由于经验不足,特征选取不合理而导致的准确率不高等问题.

本文主要致力于研究基于深度学习自然语言处理技术的智能化软件安全评估方法.我们方法的基本原理在于采用了一个两阶段的训练模式:先通过深度学习将相关的自然语言模型训练完善,以便由该模型获得安全评估所需的语义编码.然后通过普通的机器学习模型来构建语义编码与评估结果的对应关系.这一两阶段的评估设计主要考虑到了现有软件安全评估训练数据量的不足,目前我们能够收集到可以用于安全性评估训练的标定数据总记录量大约为 20 万条.我们认为这一数据量足以支撑机器学习模型的训练,但不足以直接支持深度学习模型的训练.深度学习需要海量的训练数据,否则会导致模型过拟合而使得评估精度下降.因此,我们并未采用深度学习算法直接训练安全性评估模型,而是通过一个两阶段的训练模式来实现安全性评估.这一两阶段的训练模式本质上是一种半监督的学习思想,即采用非监督的方式用深度学习过程获得较为精确的语义编码,再通过常规机器学习过程将标定的训练样本引入进来.这样做的好处在于在语义编码阶段可以使用相对成熟的机器翻译语料库进行训练,由于机器翻译语料库长期以来被专门设计用于深度学习方法的研究,这些资源足

1 https://cve.mitre.org/

2 https://nvd.nist.gov/

以支撑深度学习的训练过程,并可以得到较为准确的语义编码模型.在第二阶段,我们进一步引入了常规机器学习方法来实现安全性评估.通过这样的双重训练,本文方法可以更为有效地完成安全性评估任务.

3 评估方法

本文自然语言数据驱动的智能化软件安全评估方法围绕着互联网上与待评估软件相关的自然语言数据来进行安全评估指标的预测.其整体框架如图 1 所示,用户仅须提供待评估软件的名称和版本号,本文的评估系统会自动输出该软件可能存在的安全漏洞数量,类型以及严重程度等评估指标.本文方法主要由三部分构成.包括一个基于搜索引擎的自然语言数据爬取模块,一个基于深度学习技术的语义理解模块,以及一个基于机器学习技术的安全评估指标生成模块.数据爬取模块从互联网上搜索和待评估软件相关的页面信息,并通过简单的预处理获得待评估软件相关的自然语言数据描述.深度语义理解模型通过自然语言处理技术将上一步骤获得的待评估软件描述转换成语义编码.该语义编码最终将用于训练一个基于机器学习的安全评估模型,从而获得评估结果.在本节中,我们将分别具体介绍这 3 个部分的技术细节.

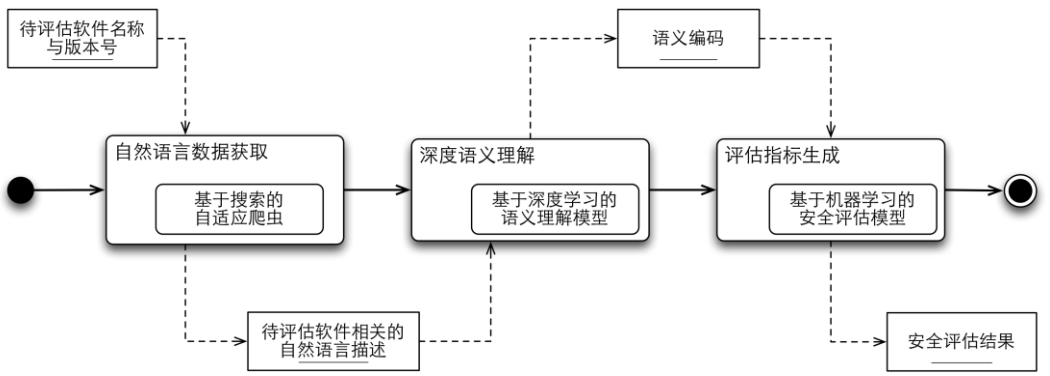

Fig.1 Main Flowchart of Natural Language Data Driven Approach

图 1 自然语言数据驱动的软件安全评估整体框架

3.1 自然语言数据获取

使用本方法进行安全评估时,用户仅需要提供软件的名称和版本号作为评估输入,如 Acrobat 10.0, Firefox 52.0.2 等等.本文方法通过基于搜索的网络爬虫自动从互联网上获取与待评估软件直接相关的自然语言描述信息,并基于这些描述信息来评估软件的安全性.在应用本文方法之前,我们并不明确指定数据源,而让系统自动在互联网上查找合适的数据.因此,本文方法引入了一个基于搜索的自适应爬虫来获取用于安全评估的自然语言数据.

本文的自适应爬虫同主流的 Web 搜索引擎(如百度网页搜索,微软必应搜索等网站)建立连接,并将用户提供的待评估软件名称和版本号作为输入来驱动搜索引擎获得有用的搜索结果.为了使评估效果更为准确,我们增加了自适应特征词来提高搜索引擎收集自然语言数据的针对性.在搜索引擎获得搜索结果后,爬虫会爬取排名靠前的搜索结果.在我们的爬虫中内置了一个广告过滤器,因此,爬虫会自动跳过搜索引擎推荐的广告结果,而直接获得搜索到的网站.

在获得搜索网址后,爬虫会自动连接对应的网站下载网页信息.对应网站可能是一个评价待评估软件的用户论坛,也有可能是待评估软件的官方描述网站,甚至是一些黑客对待评估软件的缺陷探讨等等.这些网页均由爬虫在不经过人工指定的情况下自动获取到,在极端情况下,爬虫也可能会获取到一些不相关数据,而成为后续步骤的数据噪音,这些噪音将由深度语义理解模块进行自动过滤.本文的爬虫对网页信息会做较为简单的预处理,仅保留了<body>的相关内容.并将<body>中各<input>,<id>等字段去除,并舍去标签、图片、URL 等信息,仅保留自然语言的文字段落信息作为后续步骤的评估依据.

本文通过自适应地启发式方法生成搜索特征词.搜索的初始特征词来源于现有的软件安全漏洞库,由于漏

洞库中广泛存在各软件安全漏洞的描述信息,本文的安全性评估以这些信息为重要的文字参考依据,并以此为起点在互联网上搜索相关的自然语言数据.我们对安全漏洞库中的漏洞描述信息施行文字统计,获得其词频列表,并去除虚词后以高频实词作为候选特征词输入搜索引擎.在搜索引擎返回的搜索结果中爬取用于评估的自然语言数据后,后续深度语义理解模块和评估模块会获得评估结果.同时,我们根据已知数据库中的安全数据对搜索特征词进行了反馈调整,最终我们发现,采用 2 个特征词作为搜索引擎输入采集到的自然语言数据质量较高,适合用于安全评估.

3.2 深度语义理解

由爬虫从互联网上获取待评估软件的相关自然语言数据后,这些数据将由深度语义理解模块转换成标识评估语义的向量编码.本文通过训练深度学习模型来构建安全评估的语义理解模块,深度学习是指通过增加人工神经网络的隐含层使神经网络"变深",并以大数据作为训练样本,以云计算的高性能计算能力来训练这样的人工神经网络,使训练效果得到本质的提升.许多学者指出:深度学习的效果并不明显依赖于人工设定的训练数据特征[3].由于神经网络本身可以看做由每一层神经元所代表的数学函数复合而成,神经网络的层次越深,所代表函数的复合层数就越多.在合适的训练集下,外层神经元相当于起到了自动抽取训练数据特征的作用,而使其内层的神经元所拟合的数据性质更为概括,其拟合出的数据特征也就更为高层.因此,深层的深度学习模型可以自动获得更有表达力的数据特征.

3.2.1 循环神经网络编码-解码模型

深度学习常常以不同的神经网络结构来刻画其学习过程,卷积神经网络和循环神经网络是其中最具有代表性的学习模型.在自然语言处理领域,循环神经网络使用广泛且效果显著[4],因此本文基于循环神经网络来构建自然语言数据驱动的智能化软件安全评估.与基本神经网络类似,循环神经网络也由输入层,隐含层,以及输出层构成,每一层含有一个或多个神经元作为基本的计算单位.在循环神经网络中,数据会按照一定的次序输入神经网络,而每一个时刻的输入都会使得循环神经网络的内部状态 s 发生改变.例如 $t+1$ 时刻的状态 s_{t+1} 就由该时间点的输入 i_{t+1} 和 t 时刻的状态 s_t 共同决定.即:

$$s_{t+1} = f(s_t, i_{t+1}) \tag{1}$$

为了从自然语言数据中抽取语义编码向量,我们首先构建一个面向机器翻译的循环神经网络编码-解码模型,该模型的基本结构如图 2 所示,为了表述上的清晰,我们将循环神经网络的每一个时间步横向展开.设编码部分通过了 M 个时间步完成,即将源语言的单词转换成向量后逐个输入编码部分,由公式 1 获得各个步骤的循环神经网络状态 $s_t(1 \leqslant t \leqslant M)$,最终编码部分得到的隐含层状态 s_M 即为当前源语言语句的语义编码 SC.在解码部分,循环神经网络会在每一个时间步获得目标语言单词的条件概率,且模型会从目标语言的词汇表中自动选择概率值最大的目标语言单词输出.在一个具体的时间点 t 上,解码神经网络通过当前状态 s_t 获得输出 o_t,并进一步结合语义编码 SC 来计算下一个时间步的状态 s_{t+1}.因此我们有:

$$o_t = g(s_t) \tag{2}$$

$$s_{t+1} = h(s_t, o_t, SC) \tag{3}$$

对于一个已经经过充分训练的解码模型,我们仅需要给一个语义编码 SC 和起始符号作为输入,该解码循环神经网络会自动进行 N 个时间步的迭代而输出目标语言的句子.其中每一个时间步会按照公式 2 与公式 3 的逻辑输出一个目标语言单词,直至输出到结束符号时,解码部分会自动终止.

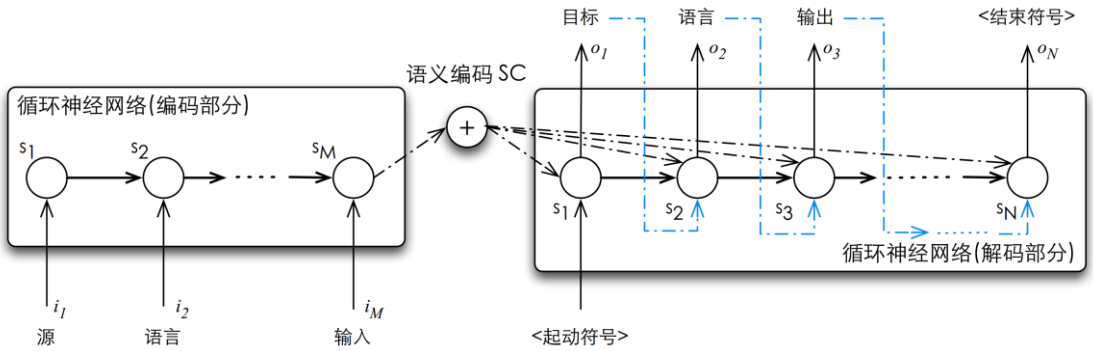

图 2 用于机器翻译的循环神经网络编码-解码模型

我们按照长短期记忆网络(Long Short-Term Memory, LSTM)的架构来构建用于安全评估的循环神经网络模型.在训练语料库充足,训练平台有足够计算能力的前提下,循环神经网络编码-解码模型规模越大,语义理解模型越为准确.我们根据现有语料库规模和主流训练平台的计算能力,构建了一个 4 隐层的循环神经网络,每个隐层 1024 个 LSTM 的神经元.

3.2.2 语义理解模型及其训练过程

本文的语义理解模块首先会利用现有语料库训练一个用于机器翻译的循环神经网络编码-解码模型,其训练过程主要由梯度下降算法对编码器和解码器做联合训练,其训练目标通过最小化公式 4 所定义的目标函数来完成[5].即通过调整循环神经网络的参数向量$\vec{\theta}$,使翻译的整体误差Error达到最小.

$$\mathrm{Error}(\vec{\theta}) = -\frac{1}{H}\sum_{x=1}^{H}\sum_{t=1}^{N}\log p_\theta(o_{tx}|i_x) \tag{4}$$

公式中 H 代表了训练集的样本数目,N 代表了每一个训练样本目标语言的最大单词数,$p_\theta(o_{tx}|i_x)$表示神经网络在模型参数$\vec{\theta}$时在第 x 个样本输入 i_x 下输出正确的第 t 个单词 o_{tx} 的概率.

本文的安全性评估并不需要像机器翻译那样将源语言转换成目标语言,而仅需要机器抽取源语言的语义向量 *SC*,这一功能仅用循环神经网络的编码模型就可以做到.因此,在进行安全性评估的时候,我们的语义理解模型会将循环神经网络编码-解码模型的编码部分和解码部分分开,而仅使用其编码模型来产生语义编码 *SC*.但在训练阶段我们仍然需要解码模型来帮助编码器通过监督学习调整到合适的参数,以产生准确的语义编码 *SC*.因此,我们基于循环神经网络编码-解码模型,利用自然语言处理领域的大量语料库来训练语义理解模型.

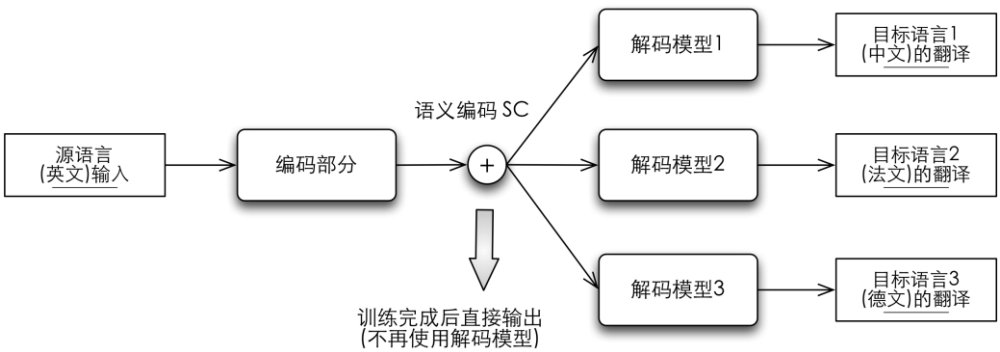

图 3 一对多的编码解码模式来训练用于安全性评估的语义理解模型

为了增强编码模型的训练效果,我们依次将多个不同的解码模型连接到同一个编码模型上进行联合训练,而每个解码器代表了机器翻译中不同的目标语言.尽管这一过程不会提高机器翻译的翻译效果,但可以增加模型中编码器的语义理解能力.其实施框架如图 3 所示,我们可以利用自然语言处理领域中相同源语言但不同目标语言的语料库来增强安全评估模型中语义理解模块的训练效果.例如,我们找到了英译中、英译法、以及英译德等不同类型的语料库,我们可以将三个完全不同的解码模型连接到同一个编码模型上,并负责各自目标语言(中文,法文与德文)的输出,而用同一个编码器来负责源语言(英文)的输入.使用这一结构的意义在于,在进行安全性评估时,会仅仅保留编码模型将源语言转换成语义编码.而多种不同语言的联合训练会使语义编码中保留更具有一般性的语义信息,而使后续的评估过程更为准确.

3.3 评估指标生成

本文方法以机器学习模型来生成最终的评估指标,这里机器学习技术在本质上通过统计模型建立了语义编码与评估指标之间的对应关系.因此,对于每一个统计指标,我们会训练一个独立的机器学习模型.软件安全性评估输出的评估指标可以分为数值型与离散型两类.例如,作为软件安全评估的一部分,漏洞数量预测将输出数值型的评估指标,即估计软件中可能会有多少个漏洞,从而帮助用户分析待评估软件遭到黑客攻击的风险有多大.而漏洞类型预测将输出离散型的评估指标,即告知用户待评估软件中可能会存在那种类型的漏洞,以帮助用户分析在软件遭受黑客攻击时可能会引起怎样的后果(不同的漏洞类型可能会引起信息泄露,数据被篡改,或者使软件拒绝服务等不同后果).

对于不同类型的软件安全评估指标,我们需要对应地构建不同类型的机器学习模型:数值型的评估指标将会采用机器学习的回归模型来生成,而离散型的评估指标我们将采用分类模型来生成.虽然从机器学习的本质来说,分类模型与回归模型具有对应关系,但因其具体工程设定的不同我们仍然构建了不同的学习模型,例如分类模型要在其输出层用 softmax 函数判断类别而回归模型一般不使用这样的设置.我们尝试了多种不同的机器学习算法,并选择准确率较高的算法来完成最终的安全性评估.

我们用于训练机器学习模型的标注数据来源于安全漏洞数据库,例如国际通用漏洞披露数据库(Common Vulnerabilities and Exposures, CVE)和美国国家漏洞数据库(National Vulnerability Database, NVD).对于这些数据库中的数据进行统计分析,我们可以得到软件名称、版本号和评估指标之间的对应关系.这些对应关系成为了我们训练评估模型的基准.我们将软件名称和版本号输入到爬虫模块和语义理解模块,获得对应的语义编码向量 SC 之后,我们将构建语义编码向量 SC 和评估指标之间的对应关系.这一对应关系将成为评估指标生成的机器学习训练数据.指标生成模型经过训练后,即可接入整体框架而完成安全性评估任务.

4 实验评估

为了评测自然语言数据驱动的智能化软件安全评估方法是否有效,我们基于国际通用漏洞披露数据库(Common Vulnerabilities and Exposures, CVE)和美国国家漏洞数据库(National Vulnerability Database, NVD)来部署本文方法的评估实验.并通过十倍交叉验证来分析本文方法的准确度.本文实验主要关注以下研究问题:

研究问题 1: 以哪一种机器学习算法构建评估指标生成模型可以更有效地完成安全性评估任务?

研究问题 2: 怎样的特征词可以使安全性评估获得更为准确的自然语言数据?

研究问题 3: 以不同的搜索引擎来构建爬虫会对评估精度有什么影响?

4.1 方法实现与实验设置

我们基于 Java 1.8 和 Scala 2.11.8 实现了本文方法的工具原型,其中深度语义理解作为一个独立模块基于 TensorFlow 3.2.1 开发,由于 TensorFlow 需要以 Python 作为开发语言,故而深度语义理解模块运行在 Python 3.5.2 的平台,评估指标生成模块是由 Scala 调用 Weka 3.8 来构建的.实验运行在一台戴尔 Precision Tower 3620 的工作站上,其 CPU 为四核的 Intel Core i7-6700 4GHz,内存为 16GB DDR4-2133.深度学习训练用了显卡加速,

工作站的显卡型号为 NVIDIA Quadro K620,显存为 2GB,并在这样的配置下进行了 128 个小时的深度学习训练来构建用于软件安全性评估的语义理解模型.

在实验中,我们以从安全数据库中总结的软件漏洞数量、漏洞类型、以及漏洞严重程度作为安全性评估指标.其中漏洞数量直接以安全数据库中对于当前软件的已知漏洞数量作为评估依据;漏洞类型采用了国际通用的 CWE(Common Weakness Enumeration)类型标识,例如 CWE-310 类型的漏洞表示待评估软件中加密算法的加密强度不够,有被破解的风险,而 CWE-824 类型是指待评估软件中存在指针的不安全访问;对于漏洞的严重程度,我们以通用漏洞评分系统(Common Vulnerability Scoring System)的漏洞评分来作为评估依据.

我们从安全数据库收集了从 1999 年 2 月 21 日至 2017 年 2 月 20 日的漏洞数据,共计 82606 个漏洞作为实验的标注数据.对于像漏洞数量与漏洞严重程度评分等数值型的安全评估指标,我们按照公式 4 计算其修正相对误差(Corrected Relative Error, CRE),并以此来判断我们评估方法的准确程度.

$$CRE = \frac{\Delta}{L+1} \times 100\% \tag{4}$$

在公式 4 中Δ 是预测值和标注值之差,L 是标注值,为了防止标注值为零(例如实际软件在安全数据库中没有漏洞报告),我们将标注值按照惯例进行了加一修正.除此之外,采用修正相对误差 CRE 的另一个重要原因是当标注值 L 较小时,相对误差很容易变的很大而难以准确反映评估的准确程度,修正相对误差能够缓解这一现象,减少预测输出结果和标注值的相关性,从而使我们对预测效果的评判更为精确.该修正相对误差所代表的实际含义为预测值误差在标注值中的比例,误差值越小则安全性评估的结果越为准确,误差值最小为零.

对于像漏洞类型这样的离散型评估指标,本文评估方法将会按照待评估软件中各类型漏洞出现的概率从高到低输出一个该指标的预测排序.我们用待评估软件中出现最多的漏洞类型在预测排序中的排名差来评估本文方法的准确程度.例如:在预测排序的第一位就是待评估软件中实际出现最多的漏洞类型,则排名差为 0,如果待评估软件中实际出现最多的漏洞类型出现在预测排序中的第二位,则排名差为 1,以此类推排在第三位的排名差为 2,排在第四位的排名差为 3.排名差体现了预测类型在概率排名上的误差值,排名差越小则在漏洞类型上的评估越为准确,与数值型评估指标的相对误差一样,排名差最小也为零.

4.2 实验1: 评估指标生成算法比较

本文方法适用于评估已发布、并在互联网上积累了一定用户评价的软件系统,对于不在 CVE 和 NVD 数据库中的相关软件,我们也能支持其安全评估,但缺乏对评估结果好坏的判定方法.因此,我们采用十倍交叉验证方法来比较不同的安全性评估指标生成算法,即将依据漏洞数据库产生的标注数据集分成 10 份,每次轮流用其中 9 份数据来训练评估指标的生成模型,并用剩下的 1 份数据进行测试,最终将 10 次的测试结果计算平均值.

我们尝试使用不同的机器学习算法来获得软件的安全评估指标,对于漏洞数量和漏洞严重程度的评分这些数值型评估指标,我们尝试了 4 种不同的机器学习回归算法来进行评估,其结果如表 1 所示,而对于漏洞分类这样的离散型评估指标,我们尝试了 3 种不同的机器学习分类算法来进行评估,其结果如表 2 所示.由于随机森林法既可以作为回归模型又可以作为分类模型,故我们对数值型评估指标和离散型评估指标均用该方法进行了尝试.

从表 1 的数据结果可以看出,不合适的机器学习算法对于漏洞预测与安全性评估几乎是无效的.在回归模型中,线性回归和多层感知器对漏洞数量预测的误差很大,特别是线性回归模型,原因在于自然语言的语义结构常常和评估结果并不构成线性关系,使用线性模型不能合理的表现出自然语言语义结构中的复杂关系,即使经过充分地学习与训练,线性回归模型也不能胜任基于自然语言的软件安全性评估任务.多层感知器不仅预测误差较大,并且训练时间很长,这是因为多层感知器是一种对数据量和训练时间要求较高的算法.目前数据库中漏洞数量与漏洞严重程度的标定训练数据还不能将多层感知器的能力完全发挥出来.M5 模型树和随机森林法是较为合适的评估算法,能得到很好的回归效果,这是因为 M5 模型树和随机森林法本质上都集成了类似于决策树的预分类过程,这一过程能很好的将语义信息归类到回归输出的大致范围,再通过进一步的回归算法,使评估结果更为精确.从实验结果来看,随机森林法比 M5 模型树需要稍多的训练时间,但评估精度更高.

事实上,在 CVE 和 NVD 等数据库中,不同软件的实际漏洞数量差别很大,有些软件目前仅被发现了几个漏洞,而另一些软件有数百个漏洞被收录在数据库中.经过对实验结果数据的进一步追踪,我们发现漏洞预测的相对误差和软件漏洞的绝对数量之间存在一定的依赖关系:当漏洞绝对数量特别小,即数据库中披露的漏洞很少时,预测相对误差会很大,而当漏洞数量较大,数据库中披露的漏洞较多时,预测相对误差会趋向于稳定,此时相对误差能客观地反映评估效果.例如,当某软件实际仅存在 1 个漏洞时,如果预测值为存在 3 个漏洞,实际仍然是较为准确的预测,但相对误差却为 200%,看起来误差很大.而当实际软件的漏洞绝对数量很大时,预测相对误差能够较为实际地反映评估效果.

表 2 的分类结果与回归结果类似,随机森林法获得了最好的分类效果,这说明在安全性评估这一具体的问题背景下,离散型评估指标与数值型评估指标具有较为相似的数据特征.因此,我们主要以随机森林法来部署后续实验,已探讨其它因素对的评估结果的影响.

表 1 对于漏洞数量、漏洞严重程度评分等数值型评估指标,采用不同回归模型的训练时间与预测误差

指标生成回归算法	漏洞数量预测算法的训练时间	漏洞数量的平均预测相对误差	漏洞严重程度评分预测的训练时间	漏洞严重程度评分的平均预测相对误差
线性回归	2.72 秒	86.78%	5.86 秒	20.03%
多层感知器	83.93 秒	44.66%	271.69 秒	12.98%
M5 模型树	0.19 秒	25.71%	0.19 秒	7.42%
随机森林法	0.22 秒	24.18%	0.38 秒	1.74%

表 2 对于漏洞类型预测,采用不同分类模型的训练时间与预测误差

漏洞分类算法	训练时间	最严重漏洞类型的预测准确率	漏洞类型排序的平均排名差
Logistic 多分类器	1.91 秒	68.81%	0.63
决策树	0.28 秒	70.25%	0.55
随机森林法	0.67 秒	82.53%	0.43

4.3 实验2: 不同特征词的评估比较

在本实验中,我们分析以怎样的特征词可以驱动本文的爬虫获得更有利于软件安全性评估的特征词.这些特征词会同软件名称和版本号一起传入 Web 端的搜索引擎,从而定位到相关自然语言信息的网站.假如没有这些特征词信息,爬虫很有可能会爬取到与安全性无关的自然语言信息,从而对评估结果造成干扰.因此,选用合适的特征词对于构建有效的安全性评估具有重要意义.

由于安全数据库中关于各个漏洞的自然语言描述是与安全性评估直接相关的数据信息,我们通过统计安全数据库漏洞描述信息中的单词词频来获取本实验的初始候选特征词.我们将描述信息中的单词按词频从高到低排列,并人工去除了词频较高的虚词等意义不明确的单词,其词频最高的前八个单词及其词频被列举在表 3 的前两列.当我们以这些高频单词来驱动软件的安全性评估时,评估效果如表 3 所示.我们通过对各指标的安全性评估效果进行打分来对特征词做整体评价.其打分规则为,当某个指标的平均预测相对误差每增加 1%,则扣 1 分;漏洞类型排序的平均排名差每相差 1 名,则扣 50 分.这样设定打分规则的原因在于,当排名差达到 2 名开外时,我们认为这一次漏洞类型排名的预测是失败的.以满分 100 分减去 3 项指标的平均扣分值,即为当前评

估效果的最终得分.

表 3 单特征词的评估效果

特征词	数据库中词频	漏洞数量的平均预测相对误差	漏洞严重程度评分的平均预测相对误差	漏洞类型排序的平均排名差	评估效果得分
Crafted	14989	24.97%	2.86%	0.62	80.39
Cross	11624	27.19%	2.66%	0.78	77.05
Vulnerability	37262	28.98%	3.55%	0.85	74.99
Attacker	61106	25.70%	4.72%	1.02	72.86
Cve	30436	28.17%	4.57%	1.03	71.92
Unspecified	18482	30.66%	4.00%	1.00	71.78
Arbitrary	41913	39.30%	4.39%	1.08	67.44
Denial	18235	39.26%	4.63%	1.13	66.54

表 4 双特征词的评估效果

双特征词组合	漏洞数量的平均预测相对误差	漏洞严重程度评分的平均预测相对误差	漏洞类型排序的平均排名差	评估效果得分
Cross, crafted	23.06%	1.07%	0.33	86.46
Cross, denial	24.42%	1.23%	0.34	85.78
Cross, vulnerability	24.74%	1.13%	0.37	85.21
Cve, cross	24.96%	1.93%	0.41	84.20
Vulnerability, denial	24.93%	2.87%	0.43	83.57
Cve, vulnerability	24.77%	2.07%	0.47	83.22
Cve, denial	25.15%	2.70%	0.46	83.05
Cve, crafted	25.41%	2.66%	0.50	82.31

表 5 三特征词的评估效果

三特征词组合	漏洞数量的平均预测相对误差	漏洞严重程度评分的平均预测相对误差	漏洞类型排序的平均排名差	评估效果得分
Cross, crafted, denial	27.46%	2.07%	0.34	84.49
Cross, crafted, vulnerability	27.74%	2.07%	0.37	83.90
Cve, cross, vulnerability	26.91%	1.89%	0.40	83.73
Cross, denial, vulnerability	27.80%	2.31%	0.37	83.80
Cve, cross, crafted	26.76%	1.88%	0.42	83.45
Cve, cross, denial	27.33%	2.00%	0.41	83.39
Cve, crafted, vulnerability	27.22%	2.50%	0.45	82.59
Cve, crafted, denial	26.72%	2.68%	0.47	82.37

表 3-5 给出了不同特征词及其组合下驱动安全性评估的预测误差,为了描述上的清晰,我们按照评估效果得分从大到小的次序来排列表中的数据.其中表 3 给出了使用单个特征词来驱动的安全性评估效果,表 4 给出了同时将两个特征词来驱动搜索引擎进行安全性评估的效果,表 5 给出了使用 3 个特征词组合来评估软件安全性的效果.从表 3-5 的评估数据可以看出,使用两个特征词来驱动安全性评估,其整体效果要优于单个特征词和三个特征词,这与搜索引擎的特点相关.一般来说,当用户使用搜索引擎来获得有用信息时,关键词数量适中最为有效.对于本文的安全性评估方法在使用 2 个特征词时,对应的搜索引擎关键词约为 4 个左右,这一关键词数目使爬虫有更高的概率获得有用信息.并由表 4 数据可知,当我们组合使用 cross,crafted 这两个特征词时评估效果最好,这两个词也正好是表 3 中评估得分最高的两个特征词.由此可见,组合使用评估较高的特征词,仍然可以获得较好的评估效果.因此在后续实验中,我们以 cross,crafted 作为特征词来获得自然语言数据.

我们在训练完成后,对相关特征词在爬虫阶段获得的数据源进行了人工追踪,追踪结果显示:爬虫以 cross,crafted 作为特征词自动从互联网上获得的自然语言数据中,70%以上的段落来源于对应软件的 Bug 汇总库、漏洞论坛或者对应软件已有的评估报告中,因此我们认为这三类数据源对于评估结果是最为直接有效的.

4.4 实验3: 搜索引擎对评估结果的影响

我们尝试采用不同搜索引擎来获得本文方法所需的自然语言数据.并观察评估效果,表 6 显示了必应, 雅虎, 百度这三个搜索引擎的评估效果,从表 6 数据可知,不同搜索引擎对评估结果的影响不大,这说明大部分搜索引擎获取安全性评估相关信息的能力是类似的.相对来说百度搜索在软件安全性评估上的精度略低于雅虎和必应.

表 6 使用不同搜索引擎时模型的预测精度

搜索引擎	漏洞数量的平均预测相对误差	漏洞严重程度评分的平均预测相对误差	漏洞类型排序的平均排名差	评估效果得分
微软必应	25.44%	1.96%	0.43	83.70
雅虎搜索	25.43%	2.27%	0.46	83.10
百度搜索	25.91%	2.40%	0.61	80.40

4.5 实验小结

由以上三组实验的观测,我们可以得出以下结论:对于研究问题 1,随机森林算法构建的机器学习模型可以更有效地完成安全性评估任务,因其决策树分类与各树的回归均值算法既能很好的将语义信息归类到回归输出的大致范围,又能更为精细的构建分段的回归模型.对于研究问题 2,我们发现组合两个评估结果较好的特征词可以获得更为有效的自然语言数据,因为这样会使搜索引擎的关键词数量适中.对于研究问题 3,我们发现不同搜索引擎对评估结果的影响不是很大,这说明大部分搜索引擎获取安全性评估相关信息的能力是类似的.

5 相关工作

本文给出了一种由自然语言数据驱动的智能化软件安全评估方法来为已发布软件的新用户提供有效地安全性评估.已有的相关工作并未涉及到这一特定需求,而大多是面向未发布或最新发布软件来进行安全性评估,因此本文针对这一应用场景提出了一种新的方法.本节主要探讨和本文的安全性评估密切相关的技术,主要包括基于代码的安全指标分析技术、以及基于机器学习的恶意软件检测技术.值得注意的是,由于关注层面的不同,这两项技术实际并不冲突.基于代码的技术主要关注底层代码的行为特征,而机器学习方法所关注的内容往

往更为高层,一般用来判定较为复杂的软件安全性行为.对于同时采用机器学习制导并基于代码的相关工作,我们根据其主要贡献一般将其归类在机器学习制导的相关技术中.

基于代码的软件安全指标分析

基于代码的软件安全指标分析技术通过对源代码或二进制码进行分析和测试,分析软件中的安全漏洞以及相应的安全性评估指标.Lund 等学者提出了 CORAS 方法通过驱动模型来获得安全评估所需的评估指标.他们利用定制的 UML-profile 来表示资产风险、安全漏洞、安全威胁和安全解决方案,并围绕着安全漏洞相互关联以及安全漏洞组合所引发的威胁来分析软件的安全性[6].Goseva-Popstojanova 等学者提出了一种基于 UML 模型的安全风险评估指标,该方法首先分析体系结构中构件与连接器的风险因子,并基于事故的可能性来评估这些因子的严重性,然后它通过马尔科夫链来计算每个场景的风险因子,并最终以场景为基础来评估用例和整个软件的安全性[7].Yacoub 等学者进一步针对软件体系结构发展了软件构件和连接器的启发式风险因子,并引入依赖图来表示场景下软件构件、连接器和构件的交互概率,再通过聚合算法对所有场景下风险因子进行合成得到系统的整体风险因子[8].Cho 等学者给出了一种针对代码混淆技术的安全评估策略,主要用于评估 Android 平台物联网软件抵御安全性攻击的能力[9].Nostro 等学者提出了多项指标,以对安全攻击的防范措施进行人工评估[10].Tang 等学者探讨了模型驱动架构(Model Driven Architecture, MDA)的安全性问题,并对模型驱动架构进行了扩展,使其支持代码级的安全评估过程[11].Wang 等学者介绍了在网络设备软件中将安全功能评估与漏洞评估相结合的集成方法[12].这些评估方法能够覆盖较多的软件安全问题,评估结果相对准确,但它们都是相对重量级的方法,需要耗费大量的人工成本,这是已发布软件的新用户所不愿承担的,因而并不适用于已发布软件的安全评估.

Schmeelk 等学者研究了如何基于静态程序分析技术对移动平台的恶意攻击进行检测[13].Rashidi 等学者对 Android 平台可能存在的安全威胁及其防护方法进行了全面地分析与评价[14].Neuner 等学者探讨了基于动态测试分析的安全评估方法,并针对移动平台对比了现有的动态分析平台及沙箱平台[15].古天龙等学者分析了分类算法中的隐私泄露问题,并针对该问题提出了基于代码的安全评估方法[16].Shabtai 等学者对谷歌的安全机制进行了人工评估,生成了易于理解的安全评估报告[17].Liu 等学者对已有的多个安全数据库进行二次统计分析,并综合构建了一个软件漏洞评估数据仓库 VRSS[18].该仓库对在数据库中目前已知的软件漏洞进行了重新打分评估.一年后,Liu 等学者进一步改进了该数据仓库,他们基于层次分析方法使软件安全漏洞的标识优先级指标更为精确[19].Bagheri 等学者提出了基于静态分析技术构造多个应用对资源访问以及应用间相互通讯的形式化模型,再利用模型验证技术分析通过多个应用协同访问未授权资源的不安全行为[20].这些基于代码的攻击检测与安全指标分析主要关注底层代码的行为特征,常常针对特定的安全性问题.与这些工作不同,本文方法通过理解 Web 上收集到的待评估软件的大量信息,自动提炼与安全相关的内容,这些内容可以是基于源代码的分析和测试的结果,也可以是用户报告的错误、问题和使用经验.它覆盖了比单纯源代码分析和测试结果更多的方面,因而可以提供更加全面的评估结果.因此本文方法更适合为已发布软件的新用户提供有效地安全性评估.

基于机器学习方法的恶意软件检测技术

基于机器学习方法的恶意软件分析技术通常提取相关的程序结构或者是程序行为作为特征,许多方法也是以基于代码的软件安全指标分析为基础的.更进一步,它们基于这些特征通过分类算法构造量化指标,最后使用机器学习算法对异常行为和正常行为进行分类和识别.Gascon 等学者研究了运用机器学习中的分类算法对函数调用图进行分析,从而可以判定运行时刻的函数调用轨迹与行为特征图之间的相似性,进而识别出恶意行为的方法[21].Afonso 等学者提出了将应用调用 API 方法的次数和调用系统方法的次数作为识别恶意软件的特征,利用分类算法根据这两个特征将恶意软件与正常软件进行区分[22].Yang 等学者将触发安全敏感行为的上下文环境与安全敏感行为一起作为识别安全漏洞的特征,再利用支持向量机基于这些特征识别软件的恶意行为,这一方法可以有效处理恶意软件将恶意行为伪装成为正常行为从而逃过安全漏洞分析[23].Gorla 等学者首先根据应用的描述信息按照用途将应用分为不同的簇,然后对属于同一簇的应用对安全敏感 API 的调用进行统计,最后利用非监督的单类支持向量机识别同一簇中的非安全行为[24].与本文的工作不同,目前这些基于机器学习

的方法主要关注利用机器学习的特点检测软件的恶意行为,而并非获得安全评估特征.

自然语言处理技术也被引入到恶意软件检测中,以对软件中自然语言描述的文档和配置信息进行分析.Lu等学者利用自然语言处理技术对 App 的描述文档进行分析,以识别出类似功能软件簇,通过簇内同行软件投票提高识别隐私信息泄露行为的准确率[25].Qu 等学者运用自然语言处理工具自动提取软件描述中声明的安全权限,构造软件运行所需安全权限模型[26].Nan等学者基于自然语言处理将Andriod App布局文件中的关键文本进行提取, 以帮助后续过程对可能泄露敏感信息的用户输入展开分析[27].Huang 等学者开发了移动应用 UI 自动分析工具,通过对 UI 上的输入域标签与提示进行理解,找出可能导致用户隐私泄露的界面元素[28].上述工作中,自然语言处理技术处理的对象是软件或者是软件的配置信息,而我们的工作中,处理的是与软件安全直接相关的自然语言描述.

6 结束语

软件安全性是衡量软件能否抵御恶意攻击的重要性质.在当前互联网环境下,黑客攻击无处不在,且手段多样,因而估计软件中可能含有的漏洞数量与类型,即对软件进行安全评估,变得十分必要.基于安全评估结果,用户可以预测软件中存在的潜在安全威胁,并针对这些威胁做好必要防范措施,以避免或减少因为软件安全问题而造成损失.在实际中用户不仅需要对未发布、或者最新发布的软件实施安全性评估,对已发布软件也会有一定的安全评估需求,例如当用户需要从市场上互为竞争的多款软件中作出选择,就会希望能够花费较低成本、较为客观地对这些软件进行第三方的评估与比较.已有的安全性评估方法往往并不适用于这一评估场景.首先,用户往往获取不到这些待评估软件的源代码,甚至取得的二进制码也是经过代码混淆的,很难部署基于代码的分析技术来提取与确认所需的评估指标.其次,随着软件规模的增长与功能逻辑日益复杂,程序分析与软件测试的运行成本也随之提高.本文提出了一种由自然语言数据驱动的智能化软件安全评估方法,该方法自适应地爬取用户在软件使用过程中对软件的自然语言评价数据,并利用深度学习方法与机器学习评估模型的双重训练来获得软件的安全性评估.由于本文的自适应爬虫能够在反馈中调整特征词,并结合搜索引擎来获得异构数据,因而可通过采集广泛的自然语言数据来进行安全评估.另外,使用一对多的机器翻译训练能有效解决将自然语言数据转换为语义编码的问题,使得用于安全评估的机器学习模型可以建立在自然语言的准确语义特征基础上.我们进一步在国际通用漏洞披露数据库(CVE)和美国国家漏洞数据库(NVD)上对本文方法进行了实验,结果显示,随机森林方法在自然语言数据驱动的智能化软件安全评估中表现突出,采用双特征词来结合搜索引擎爬取自然语言数据可以达到最低的预测误差,而使用不同的搜索引擎在预测效果上差别不大.实验结果进一步说明了本文方法在评估软件漏洞数量,漏洞类型,以及漏洞严重程度上的有效性.

References:

[1] Symantec. Internet Security Threat Report. 2012:52. https://www.symantec.com/content/dam/symantec/docs/reports/istr-17-2012-en.pdf.

[2] Symantec. Internet Security Threat Report. 2017:77. https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf.

[3] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015,521(7553):436-444.

[4] Mikolov T, Karafiát M, Burget L, Cernockỳ J, Khudanpur S. Recurrent neural network based language model. In: Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH' 10). 2010. 3.

[5] Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP' 14). Doha, Qatar: Association for Computational Linguistics, 2014. 1724-1734.

[6] Lund MS, Solhaug B, Stølen K. A Guided Tour of the CORAS Method. In: Model-Driven Risk Analysis: The CORAS Approach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. 23-43.

[7] Goseva-Popstojanova K, Hassan A, Guedem A, Abdelmoez W, Nassar DEM, Ammar H, Mili A. Architectural-level risk analysis using UML. IEEE Transactions on Software Engineering, 2003,29(10):946-960.

[8] Yacoub SM, Ammar HH. A methodology for architecture-level reliability risk analysis. IEEE Transactions on Software Engineering, 2002,28(6):529-547.

[9] Cho T, Kim H, Yi JH. Security Assessment of Code Obfuscation Based on Dynamic Monitoring in Android Things. IEEE Access, 2017,5:6361-6371.

[10] Nostro N, Matteucci I, Ceccarelli A, Santini F, Di Giandomenico F, Martinelli F, Bondavalli A. A Multi-criteria Ranking of Security Countermeasures. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. New York, NY, USA: ACM, 2016. 530–533.

[11] Tang X, Shen B. Extending Model Driven Architecture with Software Security Assessment. In: 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement. 2009. 436-441.

[12] Wang X, Shi H, Huang TYW, Lin FC. Integrated Software Vulnerability and Security Functionality Assessment. In: The 18th IEEE International Symposium on Software Reliability (ISSRE '07). 2007. 103-108.

[13] Schmeelk S, Yang J, Aho A. Android Malware Static Analysis Techniques. In: Proceedings of the 10th Annual Cyber and Information Security Research Conference. New York, NY, USA: ACM, 2015. 5:1–5:8.

[14] Rashidi B, Fung C. A Survey of Android Security Threats and Defenses. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 2015,6:3–35.

[15] Neuner S, van der Veen V, Lindorfer M, Huber M, Merzdovnik G, Mulazzani M, Weippl E. Enter Sandbox: Android Sandbox Comparison. arXiv:1410.7749 [cs], 2014,

[16] Gu T, He Z, Chang L, Xu Z. Secure Evaluation of Classification Algorithms Based onSymbolic ADD and Linear Multi-Branching Program. Chinese Journal of Electronics, 2014,(5):940-947.

[17] Shabtai A, Fledel Y, Kanonov U, Elovici Y, Dolev S, Glezer C. Google Android: A Comprehensive Security Assessment. IEEE Security Privacy, 2010,8:35-44.

[18] Liu Q, Zhang Y. VRSS: A new system for rating and scoring vulnerabilities. Computer Communications, 2011,34:264-273.

[19] Liu Q, Zhang Y, Kong Y, Wu Q. Improving VRSS-based vulnerability prioritization using analytic hierarchy process. Journal of Systems and Software, 2012,85:1699-1708.

[20] Bagheri H, Sadeghi A, Garcia J, Malek S. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. IEEE Transactions on Software Engineering, 2015,41(9):866-886.

[21] Gascon H, Yamaguchi F, Arp D, Rieck K, Structural detection of android malware using embedded call graphs[C]. //Proceedings of the 2013 ACM workshop on Artificial intelligence and security, Berlin, Germany, Berlin, Germany, 2013.

[22] Afonso VM, de Amorim MF, Grégio ARA, Junquera GB, de Geus PL. Identifying Android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques, 2015,11(1):9-17.

[23] Yang W, Xiao X, Andow B, Li S, Xie T, Enck W, AppContext: differentiating malicious and benign mobile app behaviors using context[C]. //Proceedings of the 37th International Conference on Software Engineering - Volume 1, Florence, Italy, 2015.

[24] Gorla A, Tavecchia I, Gross F, Zeller A, Checking app behavior against app descriptions[C]. //Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014.

[25] Lu K, Li Z, Kemerlis VP, Wu Z, Lu L, Zheng C, Qian Z, Lee W, Jiang G. Checking More and Alerting Less: Detecting Privacy Leakages via Enhanced Data-flow Analysis and Peer Voting. In: Proceedings of the 2015 Network and Distributed System Security(NDSS 2015). 2015.

[26] Qu Z, Rastogi V, Zhang X, Chen Y, Zhu T, Chen Z, AutoCog: Measuring the Description-to-permission Fidelity in Android Applications[C]. //Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, Arizona, USA, 2014.

[27] Nan Y, Yang M, Yang Z, Zhou S, Gu G, Wang X. UIPicker: User-Input Privacy Identification in Mobile Applications. In: 24th USENIX Security Symposium (USENIX Security 15). Washington, D.C.: USENIX Association, 2015. 993–1008.

[28] Huang J, Li Z, Xiao X, Wu Z, Lu K, Zhang X, Jiang G. SUPOR: Precise and Scalable Sensitive User Input Detection for Android Apps. In: 24th USENIX Security Symposium (USENIX Security 15). Washington, D.C.: USENIX Association, 2015. 977–992.

附中文参考文献:

[16] 古天龙, 何仲春, 常亮, 徐周波. 基于符号 ADD 和线性多分支程序的分类算法安全评估. 电子学报, 2014,(5):940-947.