开源软件漏洞检测的混合深度学习方法

李元诚[1],崔亚奇[1],吕俊峰[2],来风刚[2],张 攀[2]
1.华北电力大学 控制与计算机工程学院,北京 102206
2.国家电网公司信息通信分公司,北京 100761

摘 要:针对开源软件代码质量参差不齐和存在安全隐患的问题,提出一种基于混合深度学习模型(DCnnGRU)的开源软件漏洞检测方法。以漏洞库中的关键点为切入点构建控制流图,从静态代码中提取出与关键点存在调用和传递关系的代码片段,将代码片段数字化为固定长度的特征向量,并作为DCnnGRU模型的输入。该模型用卷积神经网络(Convolutional Neural Network,CNN)作为与特征向量交互的接口,门控循环单元(Gated Recurrent Unit,GRU)嵌入到CNN中间,作为捕获代码调用关系的门控机制。首先进行卷积和池化处理,卷积核和池化窗口对特征向量进行降维。其次,GRU作为中间层嵌入到池化层和全连接层之间,能够保留代码数据之间的调用和传递关系。最后利用全连接层来完成归一化处理,将处理后的特征向量送入softmax分类器进行漏洞检测。实验结果验证了DCnnGRU模型比单独的CNN和RNN模型有更高的漏洞检测能力,准确率比RNN高出7%,比CNN高出3%。

关键词:开源软件;漏洞检测;深度学习;卷积神经网络;门控循环单元
文献标志码:A **中图分类号:**TP391 doi:10.3778/j.issn.1002-8331.1809-0076

李元诚,崔亚奇,吕俊峰,等.开源软件漏洞检测的混合深度学习方法.计算机工程与应用,2019,55(11):52-59.
LI Yuancheng, CUI Yaqi, LV Junfeng, et al. Combined deep learning method for open source software vulnerability detection. Computer Engineering and Applications, 2019, 55(11):52-59.

Combined Deep Learning Method for Open Source Software Vulnerability Detection

LI Yuancheng[1], CUI Yaqi[1], LV Junfeng[2], LAI Fenggang[2], ZHANG Pan[2]
1.School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China
2.State Grid Information & Telecommunication Co, Beijing 100761, China

Abstract: Aiming at the problem of uneven quality or security risks of open source software, this paper proposes an open source software vulnerability detection method based on hybrid deep learning model(DCnnGRU). In this paper, the control flow graph is constructed with the key points in the vulnerability library as the entry point, and the code segment with the call and transfer relationship with the key point is extracted from the static code, and the code segment is digitized into a fixed length feature vector and used as the input of the DCnnGRU model. The model uses the Convolutional Neural Network(CNN) as an interface to interact with the feature vector. The Gated Recurrent Unit(GRU) is embedded in the middle of the CNN as a gating mechanism for capturing code call relationships. The DCnnGRU model first performs convolution and pooling processing, and the convolution kernel and the pooling window perform dimensionality reduction operations on the vector. Secondly, the GRU is embedded as an intermediate layer between the pooled layer and the fully connected layer, and can retain the call and transfer relationships between code data. Finally, the full connection layer is used to complete the normalization process, and the processed feature vector is sent to the softmax classifier for classification, and the output result is obtained. The experimental results verify that the DCnnGRU model has higher vulnerability detection capability than the CNN and RNN models alone. The accuracy rate is 7% higher than RNN and 3% higher than CNN.

Key words: open source software; vulnerability detection; deep learning; Convolutional Neural Network(CNN); Gated Recurrent Unit(GRU)

基金项目:国家电网公司总部科技项目(No.SGFJXT00YJJS1800074)。

作者简介:李元诚(1970—),男,博士,教授,博导,研究领域为机器学习与信息安全,E-mail:yuancheng@ncepu.edu.cn;崔亚奇(1995—),女,硕士研究生,研究领域为深度学习;吕俊峰(1974—),男,高级工程师,研究领域为计算机应用;来风刚(1971—),男,高级工程师,研究领域为计算机科学与技术;张攀(1989—),男,工程师,研究领域为计算机应用。

收稿日期:2018-09-06 **修回日期:**2018-10-22 **文章编号:**1002-8331(2019)11-0052-08
CNKI**网络出版:**2018-12-17, http://kns.cnki.net/kcms/detail/11.2127.tp.20181213.1805.018.html

1　引言

随着IT行业软件开发技术的不断发展,开源软件逐渐成为各大IT行业的关注热点。开源软件给开发者提供可以直接利用的代码资源,减少了重复性的工作。但由于开源社区数据规模巨大并且入门门槛很低,导致许多开源代码的质量不高,存在较多的安全隐患[1]。从2015年到2017年的两年多时间中,360安全团队从GitHub、Sourceforge等代码网站和开源社区中选出了2 228个使用范围较广的开源项目进行检测,涉及多种开发语言,包括C、C++、C#和JAVA。检测代码总量257 835 574行,发现的代码缺陷有2 626 352个。因此,对开源软件进行漏洞检测对提高开源代码质量和安全性至关重要。

传统的漏洞分析分为静态分析、动态分析和动静结合分析三种方法[2]。文献[3]比较了静态分析方法和其他程序分析方法,发现静态分析方法在检测软件漏洞时具有更高自动化程度和更快速度,但静态分析方法普遍存在误报率高的问题。文献[4]提出了一种基于动态污点分析的漏洞检测技术,实现了基于控制流和数据流的污点传播过程,但频繁的污点标记检测占用了大量内存,降低了系统性能。文献[5]提出了污染传播模型的代码静态分析和净化单元动态检测相结合的方法,用于发现web应用中的漏洞,但该方法仅用于跨站脚本攻击问题,对其他的漏洞问题检测能力较差。文献[6]提出了采用SVM分类器标记可疑代码的工具VccFinder,该工具虽然降低了误报率,但在检测不同语言代码时,每次都需要重新提取特征和进行模型训练。文献[7]开发了VulPecker工具,该工具在检测代码克隆的漏洞时误报率极低,但是不适合处理其他类型的漏洞。

近期也存在采用机器学习来实现漏洞检测的相关工作。文献[8]中提出了基于深度学习的Android恶意应用检测,采用循环神经网络对smali静态代码进行检测,但该方法只针对恶意的应用程序攻击问题,并不能发现代码本身存在的漏洞问题。文献[9]提出一种改进的长短期记忆网络(LSTM)模型,应用于开源代码的漏洞检测问题,但该模型只针对C/C++的源代码问题,并且只能处理API和库函数的调用问题。

本文在上述工作的基础上,进一步研究了开源软件源代码的漏洞特征提取过程,提出一种混合深度学习模型(DCnnGRU)来实现开源软件漏洞检测。该模型结合了CNN的快速处理数据的能力和GRU捕获时序调用关系的能力,既能快速实现程序漏洞的检测,又能保留代码之间的调用关系。实验验证了本文提出的DCnnGRU模型比现存的其他漏洞检测模型拥有更高的漏洞检测能力。

2　开源软件代码特征提取

开源软件源代码的特征提取是漏洞检测的关键。首先以漏洞库中的关键点为切入点进行程序切片,从开源代码中提取包含漏洞特征的代码片段,称这些代码片段为"代码单元集"[8]。其次将包含漏洞的代码单元集向量化处理,把特征表示成深度学习模型能够处理的向量形式。开源代码的特征处理流程如图1所示。

2.1　漏洞库

为了确保切片工具准确定位到包含漏洞特征的代码部分,如:API误用的代码部分、缓存区溢出的代码部分、SQL注入的代码部分和整数溢出的代码部分等,需要设计漏洞库,在漏洞库中定义程序切片的关键点。以API误用为例,程序中API的调用函数就是漏洞库中API误用的关键点,以该调用函数作为切入点,提取出代码中与关键点有关的参数、语句和表达式。因此,开源软件漏洞库的设计是静态代码漏洞检测中一个必不可少的环节。

本文设计的开源软件漏洞库主要依据CVE漏洞数据库。CVE(Common Vulnerabilities and Exposures)兼容了28个社区和机构,收录了约6 500个条目,是目前漏洞扫描评估权威的标准漏洞库。除此之外,本文还结合了其他大型漏洞信息库,如:CWE、NVD、CNNVD等。通过对比分析,将漏洞库大体分为:输入验证、缓冲区溢出、内存管理、API误用、错误处理、信息泄露和跨站脚本这7大类漏洞问题[10]。

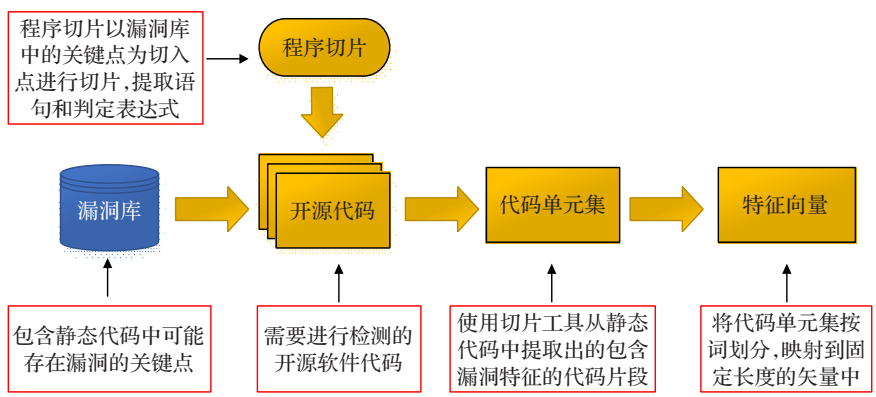

图1　静态代码特征处理流程

本文采用的7大类漏洞问题的部分关键点如表1所示。

表1 部分程序漏洞与包含的关键点

程序漏洞	漏洞库中包含的部分关键点
输入验证问题	insect,create,select,alter,update,exec,order,cookie,subject,system,command,open,close,getProperty,getRuntime
缓冲区溢出问题	strcpy,strlen,strcat,strchr,scanf,sprintf,strerror,strcoll,sbumpc,strcnpy,cin,gets,fgets,getchat,getc,getpass,malloc,istream,printf,vprintf,fprintf,memmove,recv,syslog,bcopy,fflush
API误用	cin,gets,fgets,getchat,getc,getpass,memcpy,malloc,getParameter,equals,getProperty,read,gethostbyaddr
内存管理问题	malloc,calloc,realloc,alloca,free,new,delete,memcpy,memmove,memcmp,memchr,memset,mmap,munmap,memccpy,getpagesize
错误处理问题	_alloca,catch,throw,EnterCriticalSection
跨站脚本问题	URL,submit,cookie
信息泄露问题	malloc,calloc,realloc,alloca,memcpy,memmove

2.2 程序切片

程序切片用于实现静态代码漏洞特征提取,将静态代码处理成包含特征的代码单元集。切片过程中,以漏洞库中的关键点为切入点,根据程序中函数之间相互调用的顺序和数据参数的流向,构建出控制流图和数据流图,从而提取出与关键点有关的表达式和语句,去除与特征无关的代码语句和注释[11]。目前已存在许多有关程序切片的算法和工具,本文采用LLVM来完成静态代码的切片。程序切片的示例如图2所示,红框内的"auth=strcmp(pword,PASSWORD)"是关键点。

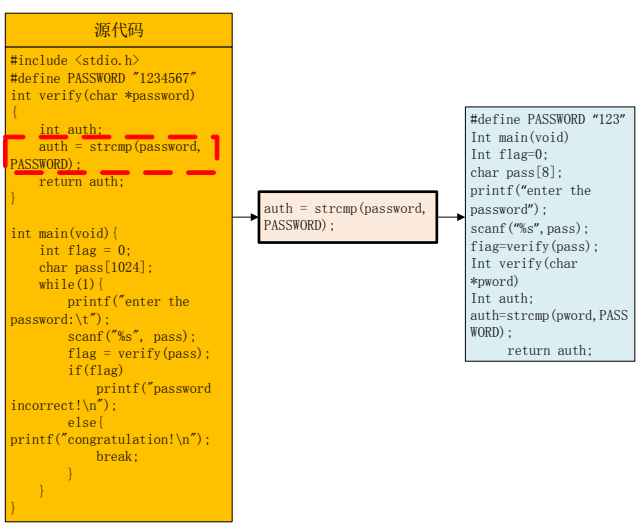

图2 程序切片示例

2.3 特征向量化表达

程序切片之后得到了包含漏洞特征的代码单元集,代码单元集并不能直接作为深度学习模型的输入,需要将其量化成固定长度的向量。本文采用词向量化模型word2vec来完成特征的向量化,word2vec模型通过构建多层神经网络处理代码单元集,处理过程中不断地改正神经网络的参数并且进行一系列线性和非线性的运算,最后得到需要的词向量。

将代码单元集向量化之前,还要对代码单元集进行规则化处理,将代码中用户自己定义的变量和函数名称一一对应的换成标准的符号名称,本文采用静态分析工具cppcheck逐行遍历代码,完成用户定义变量和标准化名称的替换。对代码单元集的标准化和向量化过程如图3所示。

3 DCnnGRU漏洞检测模型

3.1 混合深度学习模型DCnnGRU

开源软件的漏洞检测要求检测模型能够自主判断输入的代码单元集中是否包含漏洞。CNN虽然在漏洞检测时有较好的分类能力,却不能很好地保留代码语句之间的上下文关系,过于复杂的神经网络结构会随着层数增多而存在梯度消失问题[12]。RNN常用于处理时序问题,可以更好地表达代码之间的上下文调用关系,但RNN同样存在梯度消失的问题。GRU是RNN的改进模型,而且拥有比较简单的内部结构,参数较少使得内部计算更少且更容易收敛[13]。

本文提出将CNN和GRU相结合,有机融合了两种模型的优势,构建了一种更适合于开源软件漏洞检测的新模型。将CNN作为与特征向量进行交互的接口,GRU作为处理代码语句之间关系的门控机制,构成DCnnGRU模型。CNN在处理数据方面的效率与GRU相比更高更快,并且卷积核的自动学习能力也比GRU更强[14],而GRU模型不仅解决了CNN中的梯度消失问题,而且可以捕获CNN忽略的代码函数之间的调用信息。DCnnGRU模型的结构如图4所示。

在图4中,首先是CNN的卷积和池化处理,CNN能够快速处理高维数据并且在降维时最大程度上保障特征数据的不变性[15]。其次,把GRU嵌入到池化层和全连接层之间,用GRU来保留代码数据之间的上下调用关系。最后采用全连接层来完成归一化处理,将处理后的输出值送给softmax分类器进行分类检测,得到分类结果[16]。

输入层是经过预处理的50×50维的特征矩阵,图中的红色方块代表的是大小为7×7的卷积核,卷积核即为感受视野中的权重矩阵,模型中把感受视野对输入数据的扫描间距设为1。扫描到边界时可能存在出界现象,需要对边界进行扩充,出界部分的值设为0。卷积层的输入是输入层中50×50的特征矩阵,输出的矩阵维数由公式(1)确定:

图3　代码单元集的向量化过程

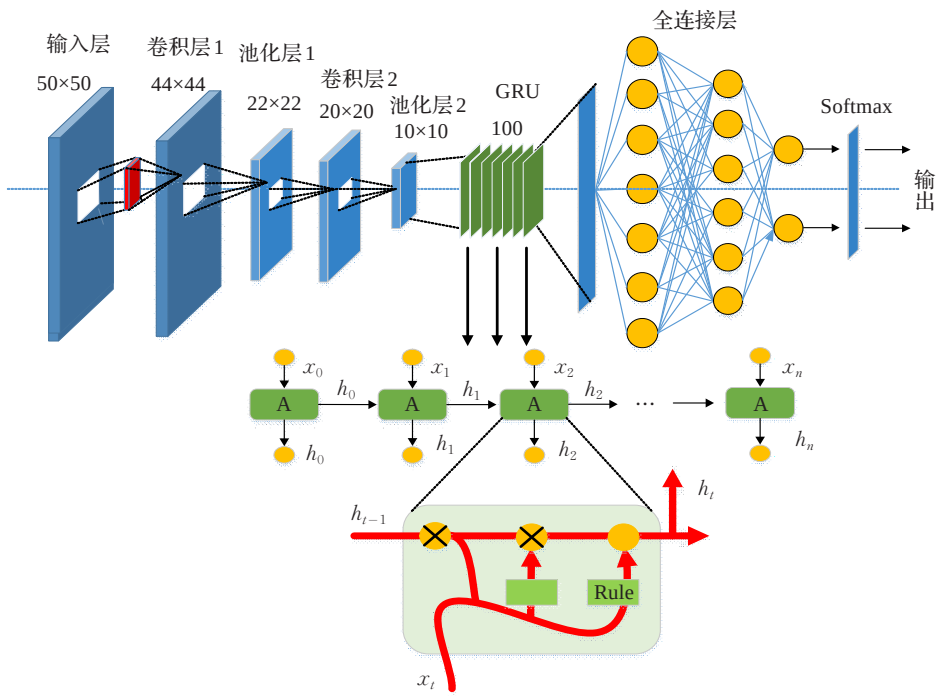

图4　DCnnGRU 模型结构

$$\begin{cases} height_{out} = \dfrac{height_{in} - height_{kernel} + 2 \times padding}{stride} + 1 \\ width_{out} = \dfrac{width_{in} - width_{kernel} + 2 \times padding}{stride} + 1 \end{cases} \quad (1)$$

公式(1)中,*height* 和 *width* 代表矩阵的长和宽,*padding* 是填充模式,*stride* 是步长,准确来说,每个卷积核还包含一个 bias 参数,但该公式省略了 bias 。在 DCnnGRU 模型模型中,*padding* 取 0,*stride* 取值为 1,输出矩阵的长和宽都为(50−7+2)/1+1=44,即池化层 1 的输入矩阵为 44×44 维。

池化层主要是完成对特征的压缩和降维,并且防止过度拟合。在池化层 1 中,采用大小为 2×2 的过滤器,步长选择为 2。能够得出池化层 1 的输出为 22×22。卷积层 2 和池化层 2 的处理过程与卷积层 1 和池化层 1 相似,

不再详细介绍。

GRU嵌入到池化层和全连接层之间。由于CNN在使用不同大小的过滤器和窗口处理数据时,往往丢失了这些代码数据之间的上下调用和传递关系,除此之外,神经网络层数过多也会存在梯度消失问题,所以需要GRU在整个模型当中充当存储时序信息和控制门的作用。

在GRU结构图中,x 作为输入,h 作为输出,f_i 是输入信息中被遗忘的部分,r_i 是输入信息中被记忆的部分。GRU中的计算如公式(2)所示:

$$\begin{cases} r_t=\sigma(W_r\times[h_{t-1},x_t]) \\ f_t=\sigma(W_f\times[h_{t-1},x_t]) \\ \tilde{h}_t=Relu(W\times[f_t\times h_{t-1},x_t]) \\ h_t=(1-r_t)\times h_{t-1}+r_t\times\tilde{h}_t \end{cases} \tag{2}$$

公式(2)中,W 表示权重,$relu$ 是激活函数,为了使DCnnGRU具有非线性建模能力,在GRU中添加激活函数。选择计算速度更快并且能够减轻梯度消失的relu函数作为激活函数,如公式(3)所示:

$$f(x)=\max(0,x) \tag{3}$$

其中 x 是输入,$f(x)$ 是输出,relu能够在 $x>0$ 时保持梯度不衰减,缓解梯度消失问题。

程序漏洞检测问题其实是一个二分类问题,存在漏洞或者没有漏洞。所以需要在模型的最后添加全连接层和softmax层。全连接层负责对特征进行进一步的降维和提纯,分类器负责最后样本中是否包含漏洞。通过滤波器(也叫卷积核),全连接层将输入和输出联系到了一起,全连接部分如公式(4)所示:

$$W*\boldsymbol{x}+\boldsymbol{b}=z \tag{4}$$

其中,$\boldsymbol{x}=[x_0\ x_1\ x_2\ \cdots\ x_n]^{\mathrm{T}}$,是输入向量;$\boldsymbol{y}=[y_0\ y_1\ y_2\ \cdots\ y_m]^{\mathrm{T}}$,是输出向量,那么滤波器部分就是一个大小为 $m\times n$ 的矩阵,$\boldsymbol{b}$ 是一个偏置项,$\boldsymbol{b}=[b_0\ b_1\ b_2\ \cdots\ b_m]^{\mathrm{T}}$。

softmax分类器广泛用于解决各领域多分类问题。softmax函数的输入特征设为 x,概率值为 $p(y=j|x)$,假设函数如下:

$$h_\theta(x)=\begin{bmatrix} p(y^{(i)}=1|x^{(i)};\theta) \\ p(y^{(i)}=2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)}=k|x^{(i)};\theta) \end{bmatrix}=\frac{1}{\sum_{j=1}^{k}\mathrm{e}^{\theta_j^{\mathrm{T}}x(i)}}\begin{bmatrix} \mathrm{e}^{\theta_1^{\mathrm{T}}x^{(i)}} \\ \mathrm{e}^{\theta_2^{\mathrm{T}}x^{(i)}} \\ \vdots \\ \mathrm{e}^{\theta_k^{\mathrm{T}}x^{(i)}} \end{bmatrix} \tag{5}$$

参数 θ 是通过训练得到的,θ 的设置需要使回归代价函数最小化,k 是向量的维数,softmax的回归代价函数为:

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}1\{y^{(i)}=j\}\mathrm{lb}\frac{\mathrm{e}^{\theta_j^{\mathrm{T}}x^{(i)}}}{\sum_{i=1}^{k}\mathrm{e}^{\theta_j^{\mathrm{T}}x^{(i)}}}\right]+\frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2 \tag{6}$$

其中 $\frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2$ 是权重衰减项,为了使 $J(\theta)$ 的值最小化,采用迭代最优算法,通过求导,可以得到梯度公式:

$$\nabla_{\theta_j}\boldsymbol{J}(\theta)=-\frac{1}{m}\sum_{i=1}^{m}[x_i(1\{y_i=j\}-p(y_i=j|x_i;\theta))]+\lambda\theta_j \tag{7}$$

$\nabla_{\theta_j}\boldsymbol{J}(\theta)$ 表示一个向量,通过最小化 $J(\theta)$,就可以实现一个softmax模型。

3.2 基于DCnnGRU模型的漏洞检测

漏洞检测过程可分为两个阶段:静态代码预处理阶段和漏洞检测模型训练测试阶段[17]。

在静态代码预处理阶段,首先是参考CVE中真实的漏洞案例,提取出案例中具体出错的API函数,将API函数分为7类构建成本文的开源软件漏洞库,以漏洞库中的API函数为关键点,作为程序切片的切入点。其次,收集数据集,采用LLVM对数据集程序进行切片,从数据集代码中提取关键点,构建关键点函数的控制流程图。在控制流图中,每个节点都是一个基本块,通过基本块的各个分支来寻找关键点函数有关的变量和操作,最后截取所有与关键点函数有关的基本块,形成300多个代码单元集。此外,还需对代码单元集进行标准化和向量化处理,使用word2vec将300多个代码单元集批量的进行向量化,得到训练样本。利用同样的方法得到测试样本。最后,将所有的特征向量归一化处理,根据样本的大小,将特征向量处理成50×50维的特征矩阵,纵坐标的大小是词向量的维数,横坐标是词向量的个数,词向量不足50个的用0填充。

在漏洞检测模型训练和测试阶段,需要编写训练模型和测试模型,整个编译过程都使用python语言。为测试样本添加标签,包含漏洞的样本标签设为“0”,不含漏洞的样本标签设为“1”。取215个样本作为训练数据,实验中采用批量抽取的方式从测试样本中每次抽取固定个数的样本,通过多次迭代训练模型。测试阶段,使用训练过的DCnnGRU模型进行测试,对比模型测试结果与实际结果是否相同,测试DCnnGRU模型的检测能力。

整个静态代码漏洞检测的整体流程如图5所示。

4 实验结果与分析

静态代码的特征处理的实验环境是内存2 GB,硬盘20 GB,一个处理器,64位的Ubuntu虚拟环境。漏洞检测模型的训练和测试环境是8 GB内存,处理器为Intel® Core™ i5-3230M,64位的Win10物理机。开源数据往往存在类不平衡的问题,即数据当中大多数是正样本(没有漏洞的样本),而负样本的个数偏低(存在漏洞的样本),这样的不平衡问题会影响漏洞检测模型的性能。为了避免类不平衡问题影响模型性能,本文实验数据来自美国国家标准与技术研究院(National Institute of Standards and Technology,NIST)官网,包含各种CVE

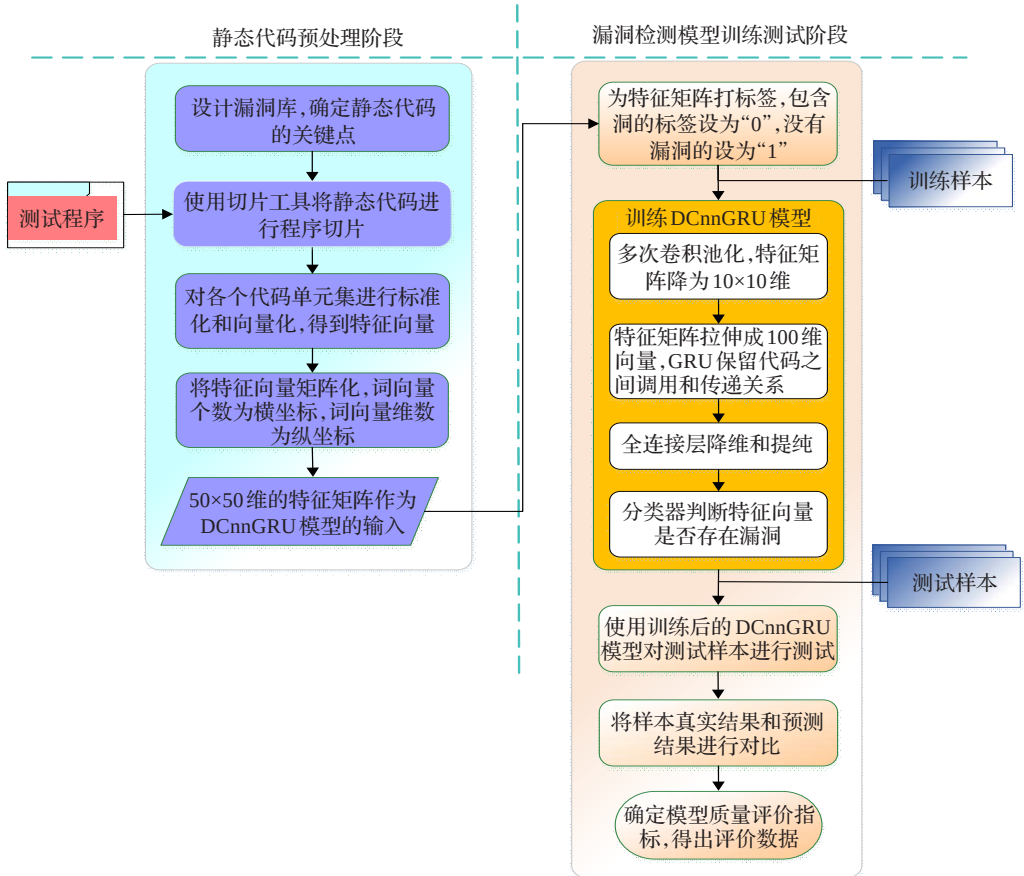

图5 静态代码漏洞检测流程

标准库中的漏洞案例。对该数据集进行预处理,包括特征提取,特征的标准化和向量化。最终形成300多个样本作为实验数据(正样本数为171个,负样本数为144),取215个样本作为训练数据(正样本117个,负样本98个)。剩下的100个作为测试样本,54个正样本,46个负样本,用于验证DCnnGRU模型的漏洞检测能力。

4.1 模型评估指标

在训练和测试模型之前,需要先给出漏洞检测模型的评估指标,精确率(ACC)和损失率(Loss)是经常采用的两个指标。参考主流的评估指标体系,根据预测结果和真实结果的区别,分为以下4种情况[18]:

TP:预测结果为正,真实结果为正;FP:预测结果为正,真实结果为负。

FN:预测结果为负,真实结果为正;TN:预测结果为负,真实结果为负。

其中准确率 ACC 的计算如公式(8)所示:

$$ACC=\frac{TP+TN}{TP+FP+FN+TN} \tag{8}$$

DCnnGRU模型的损失率由交叉熵损失函数来计算,通过计算交叉熵来反映预测结果和真实结果之间的差距。用 P 表示预测结果的集合,用 r 表示真实结果的集合,两个集合的交叉熵可以定义如下:

$$H(p,r)=E_p[-\mathrm{lb}\,r]=H(p)+D_{\mathrm{KL}}(p\|r) \tag{9}$$

其中 $H(p)$ 表示 p 的熵,$D_{\mathrm{KL}}(p\|r)$ 是KL距离,用来衡量两个集合的距离。

4.2 DCnnGRU模型的训练和测试

训练阶段,迭代3 000个训练周期,采用小批量梯度下降法算法(MBGD)进行批量抽取,每迭代10次,就输出一次当前的训练准确率(Training ACC)和训练损失率(Training Loss),并且保存模型文档。模型文档中保存着训练神经网络时调节设置的各项权重值,方便测试阶段直接加载使用。训练过程中模型的部分的检测结果如表2所示。

表2 DCnnGRU模型训练结果

迭代次数	Training ACC	Training Loss
10	0.370 6	0.663 5
20	0.622 5	0.378 2
30	0.874 2	0.237 5
60	0.901 7	0.158 1
90	0.903 2	0.154 3
120	0.903 3	0.154 1
150	0.903 3	0.154 1

训练过程中,迭代次数在500之后的准确率和损失率稳定在0.903 3和0.154 1,将此时模型的各项权重值保存在ckft模型文档中,该文档中的各项参数就是在测试阶段要使用的参数。训练过程中准确率和损失率的变化情况如图6所示。

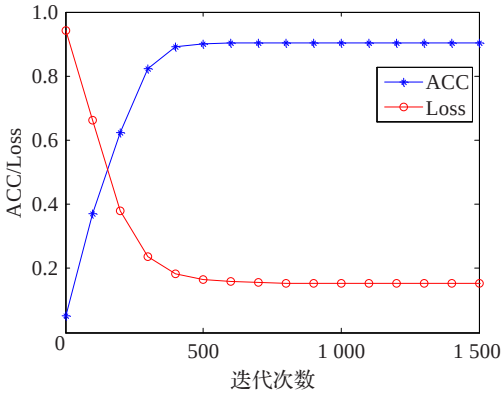

图6　DCnnGRU模型训练结果

从图6中可以看出,迭代次数小于500时,整个曲线中准确率存在明显上升现象;迭代次数大于500次时,准确率的曲线趋于平稳,保持在0.9左右。损失率在迭代次数小于500的时候,整个曲线呈现明显的下降趋势,迭次数大于500次时,曲线也同样趋于平稳,保持在0.15左右。

模型训练完毕后,对测试样本进行测试,通过加载训练时保存的模型文档,可以直接将模型恢复到训练结束时的状态。每次从315个测试样本中随机取100个样本进行测试,总共取5次。测试准确率(Test ACC)和测试损失率(Test Loss)如表3所示。

表3　DCnnGRU模型测试结果

样本	Test ACC	Test Loss
第一次	0.880 0	0.161 7
第二次	0.870 0	0.167 4
第三次	0.870 0	0.170 7
第四次	0.860 0	0.197 2
第五次	0.870 0	0.159 1

为了进一步证明DCnnGRU模型具有较高的漏洞检测能力,将DCnnGRU模型与单一的CNN、RNN以及现有漏洞检测模型VulDeePecker[9]进行比较。使用相同的数据集训练和测试CNN、RNN和VulDeePecker模型,将四个模型的测试的实验结果进行比较,具体结果如图7所示。

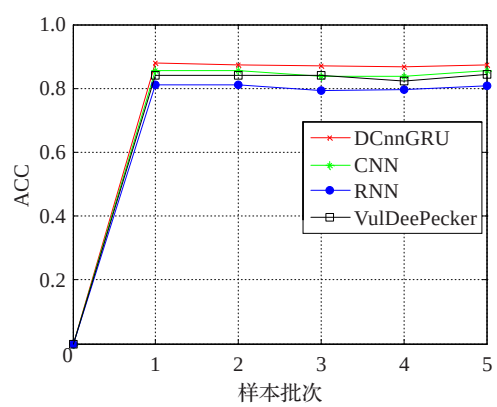

图7　深度学习模型精确度对比

图7是测试阶段中CNN、RNN、VulDeePecker和DCnnGRU模型的准确率对比。从图7中可以看出,各个模型准确率的波动都比较稳定。DCnnGRU模型的准确率明显最高,在0.87左右,CNN的准确率在0.85左右浮动,VulDeePecker的准确率在0.84左右,准确率最低的是RNN,在0.8左右变换。

图8是测试阶段中CNN、RNN、VulDeePecker和DCnnGRU模型的损失率的对比,从图8中可以看出,本文提出的DCnnGRU模型具有最低的损失率,VulDeePecker、CNN、RNN的损失率依次增高。

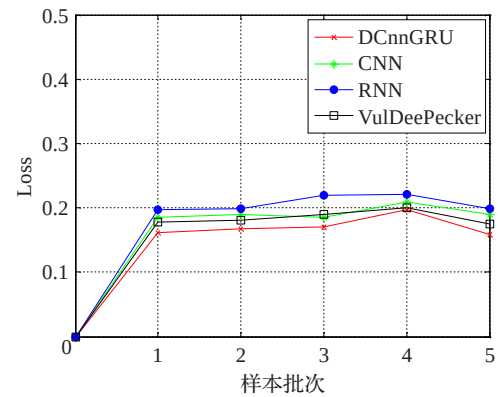

图8　深度学习模型损失度对比

将所用实验数据整理收集,可以得到表4。表中的信息是四个模型的训练数据和测试数据的平均值,可以看出DCnnGRU的实验结果最优。

表4　深度学习模型数据对比

模型	Training ACC	Training Loss	Test ACC	Test Loss
RNN	0.837 2	0.181 9	0.800 0	0.206 8
CNN	0.881 0	0.170 2	0.840 0	0.191 8
VulDeePecker	0.879 1	0.161 3	0.830 0	0.184 4
DCnnGRU	**0.903 3**	**0.154 1**	**0.870 0**	**0.171 3**

实验结果表明,将GRU嵌套到CNN的池化层和全连接层是切实可行的,本文提出的DCnnGRU模型既能在降维时最大程度保证特征向量的不变性,又能保留代码之间的调用关系,具有更强的漏洞检测能力,并且与CNN、RNN和VulDeePecker模型相比,DCnnGRU的准确率更高且损失率更低。

5　结论

本文提出一种基于混合深度学习的开源软件漏洞检测方法。先对程序进行预处理,采用程序切片技术从静态代码中提取特征,以漏洞库为标准,依据数据流程图和控制流程图提取代码单元集,将代码单元集标准化和向量化,处理成混合深度学习模型能够识别的输入形式。提出将深度学习中的CNN模型和GRU模型融合,CNN中的卷积层和池化层,能够快速高效地处理高维数据并且最大程度上保障特征数据的不变形。GRU是

在循环神经网络的基础上进一步改进的模型,它不仅能够捕获上下文代码数据之间的调用和传递关系,并且结构更加简单。DCnnGRU模型有机融合二者的优势,既能快速处理特征数据又可以保留代码之间的调用关系,与单一的CNN和RNN模型相比有更强的漏洞检测能力。

参考文献:

[1] 黄井泉.开源软件热度分析系统的研究与实现[D].长沙:国防科学技术大学出版社,2014.

[2] Shahriar H,Zulkernine M.Mitigating program security vulnerabilities:approaches and challenges[J].ACM Computing Surveys,2012,44(3).

[3] 夏一民.基于静态分析的安全漏洞检测技术研究[J].计算机科学,2006,33(10):279-282.

[4] 陆开奎.基于动态污点分析的漏洞攻击检测技术研究与实现[D].成都:电子科技大学出版社,2013.

[5] 潘古兵,周彦晖.基于静态分析和动态检测的XSS漏洞发现[J].计算机科学,2012,39(s1):51-53.

[6] Perl H,Dechand S,Smith M,et al.VccFinder:finding potential vulnerabilities in open-source projects to assist code audits[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security,2015:426-437.

[7] Li Z,Zou D,Xu S,et al.VulPecker:an automated vulnerability detection system based on code similarity analysis[C]//Conference on Computer Security Applications,2016:201-213.

[8] 陈四通.基于深度学习算法的Android恶意应用检测技术研究与实现[D].北京:北京邮电大学出版社,2016.

[9] Li Zhen,Zou Deqing,Xu Shouhuai,et al.VulDeePecker:a deep learning-based system for vulnerability detection[C]//Network and Distributed Systems Security(NDSS) Symposium,2018.

[10] Shin Y,Meneely A,Osborne J A,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.

[11] Mao Y H,Lin R Q.Research on the application of dynamic slicing technology in software testing[C]//IEEE International Conference on Computer and Communications,2017.

[12] Saxe J,Berlin K.eXpose:a character-level convolutional neural network with embeddings for detecting malicious URLs,file paths and registry keys[J].arXiv preprint arXiv:1702.08568,2017.

[13] Qu Z,Su L,Wang X,et al.A unsupervised learning method of anomaly detection using GRU[C]//IEEE International Conference on Big Data & Smart Computing,2018.

[14] Wu F,Wang J,Liu J,et al.Vulnerability detection with deep learning[C]//IEEE International Conference on Computer and Communications,2018:1298-1302.

[15] 彭天强,粟芳.基于深度卷积神经网络和二进制哈希学习的图像检索方法[J].电子与信息学报,2016,38(8):2068-2075.

[16] Su J,Tan Z,Xiong D,et al.Lattice-based recurrent neural network encoders for neural machine translation[C]//Proceedings of the 31st AAAI Conference on Artificial Intelligence,2017:3302-3308.

[17] Chernis B,Verma R.Machine learning methods for software vulnerability detection[C]//ACM International Workshop,2018:31-39.

[18] Pendleton M,Garcia-Lebron R,Cho J,et al.A survey on systems security metrics[J].ACM Comput Surv,2017,49(4).

(上接第51页)

[7] Allam D,Yousri D A,Eteiba M B.Parameters extraction of the three diode model for the multi-crystalline solar cell/module using moth-flame optimization algorithm[J].Energy Conversion and Management,2016,123:535-548.

[8] Buch H,Trivedi I N,Jangir P.Moth flame optimization to solve optimal power flow with non-parametric statistical evaluation validation[J].Cogent Engineering,2017,4(1):1286731.

[9] 李伟琨,阚波,王万良,等.基于多目标飞蛾算法的电力系统无功优化研究[J].计算机科学,2017,44(b11):503-509.

[10] Jangir N,Pandya M H,Trivedi I N,et al.Moth-flame optimization algorithm for solving real challenging constrained engineering optimization problems[C]//2016 IEEE Students' Conference on Electrical,Electronics and Computer Science(SCEECS),2016:1-5.

[11] Garg P,Gupta A.Optimized open shortest path first algorithm based on moth flame optimization[J].Indian Journal of Science and Technology,2017,9(48).

[12] 郭丽萍,李向涛,谷文祥,等.改进的萤火虫算法求解阻塞流水线调度问题[J].智能系统学报,2013,8(1):33-38.

[13] Jiao B,Lian Z,Gu X.A dynamic inertia weight particle swarm optimization algorithm[J].Chaos Solitons & Fractals,2008,37(3):698-705.

[14] Rahnamayan S,Tizhoosh H R,Salama M M A.Opposition-based differential evolution[C]//IEEE Symposium on Foundations of Computational Intelligence,2007:81-88.

[15] 喻飞,李元香,魏波,等.透镜成像反学习策略在粒子群算法中的应用[J].电子学报,2014,42(2):230-235.

[16] Soliman G M A,Khorshid M M H,Abou-El-Enien T H M.Modified moth-flame optimization algorithms for terrorism prediction[J].International Journal of Application or Innovation in Engineering and Management,2016,5:47-58.