基于CNN-GAP可解释性模型的软件源码漏洞检测方法

王 剑* 匡洪宇 李瑞林 苏云飞
(国防科技大学电子科学学院 长沙 410073)

摘 要：源代码漏洞检测是保证软件系统安全的重要手段。近年来，多种深度学习模型应用于源代码漏洞检测，极大提高了漏洞检测的效率，但还存在自定义标识符导致库外词过多、嵌入词向量的语义不够准确、神经网络模型缺乏可解释性等问题。基于此，该文提出了一种基于卷积神经网络(CNN)和全局平均池化(GAP)可解释性模型的源代码漏洞检测方法。首先在源代码预处理中对部分自定义标识符进行归一化，并采用One-hot编码进行词嵌入以缓解库外词过多的问题；然后构建CNN-GAP神经网络模型，识别出包含CWE-119缓冲区溢出类型漏洞的函数；最后通过类激活映射(CAM)可解释方法对结果进行可视化输出，标识出可能与漏洞相关的代码。通过与Russell等人提出的模型以及Li等人提出的VulDeePecker模型进行对比分析，表明CNN-GAP模型能达到相当甚至更好的性能，且具有一定的可解释性，便于研究人员对漏洞进行更深入的分析。

关键词：源代码漏洞检测；深度学习；神经网络可解释性
中图分类号：TN919.31; TN915.08 **文献标识码**：A **文章编号**：1009-5896(2022)07-2568-08
DOI: 10.11999/JEIT210412

Software Source Code Vulnerability Detection Based on CNN-GAP Interpretability Model

WANG Jian KUANG Hongyu LI Ruilin SU Yunfei
(*College of Electronic Science and Technology, National University of Defense Technology, Changsha* 410073, *China*)

Abstract: Source code vulnerability detection is an important method to ensure the security of software system. In recent years, a variety of deep learning models are applied to source code vulnerability detection, which improves greatly the efficiency of vulnerability detection. However, there are still some problems in source code vulnerability detection based on deep learning, such as too many words outside the database caused by user-defined identifier, inaccurate semantics of embedded word vector, lack of interpretability of neural network model, and so on. A new software source code vulnerability detection method is proposed based on Convolution Neural Networks (CNN) and Global Average Pooling (GAP) interpretability model. Firstly, some user-defined identifiers are normalized in the source code preprocessing, and one hot coding is used for word embedding to alleviate the problem of too many words outside the database. Then, CNN-GAP neural network model is built to identify the functions containing CWE-119 type vulnerabilities. Finally, Class Activation Mapping (CAM) interpretable method is used to output visually the results and identify the codes that may be related to vulnerabilities. Compared with the model proposed by Russell and Vuldeepecker model proposed by Li et al., the experimental results show that CNN-GAP model can achieve quite or even better performance, and has a certain interpretability, which is convenient for researchers to analyze the vulnerability more deeply.

Key words: Source code vulnerability detection; Deep learning; Interpretability of neural network

收稿日期：2021-05-12；改回日期：2021-11-11；网络出版：2021-11-15
*通信作者： 王剑 jwang@nudt.edu.cn
基金项目：国家自然科学基金(61702540)，湖南省自然科学基金(2018JJ3615)
Foundation Items: The National Natural Science Foundation of China (61702540), The Hunan Provincial Natural Science Foundation (2018JJ3615)

1　引言

在冯诺依曼体系下，软件漏洞不可避免。现代软件的规模越来越大，功能越来越复杂，导致软件漏洞层出不穷。近年来各种开源社区的兴起，使得代码共享、协作开发的编程思想蔚然成风。当前，很多企业所使用的软件都是在开源库的基础上进行开发的。因此，对源代码进行漏洞检测能尽早发现软件漏洞，保证软件系统的安全性。

传统的源代码漏洞检测方法主要分为两类。一类是基于静态分析的检测方法，包括基于逻辑推理的抽象解释[1]和模型检验[2]方法，基于抽象语法树[3]、控制流图[4]、数据流图[4]等中间表示形式的程序分析方法；另一类是动态检测方法，包括模糊测试[5]、符号执行[6]、污点分析[7]等。基于静态分析的检测方法根据人工经验和专家知识构建相应的漏洞模式进行检测，其代码覆盖率高，但准确率低、漏报高。基于动态分析的检测方法需要将源码编译成二进制文件后运行程序进行检测，其准确率高，但效率低、开销大，通常用于缺乏源码的二进制软件漏洞检测。

在大数据驱动下，深度学习技术逐渐应用于漏洞检测[8]。Lin等人[9,10]设计了一种基于Bi-LSTM神经网络模型的漏洞检测方法。该方法利用抽象语法树表示函数源码，可实现跨项目函数的迁移表示和漏洞检测。他们还设计了一种LSTM框架，实现了对异构数据源在函数粒度上的漏洞检测[11]。同时，Liu等人[12]提出了一种基于深度学习和正样本模糊生成的漏洞检测系统DeepBalance。该系统通过生成“非真实”的正样本以平衡正负样本的数量，然后利用Bi-LSTM框架进行漏洞检测，提高了漏洞检测的准确率。Liu等人[13]提出了一种跨域漏洞检测系统CD-VulD。该系统利用度量迁移学习框架，最小化源域与目标域之间的分布差异来学习跨域表示，并通过Bi-LSTM框架进行漏洞检测，大幅提高了漏洞检测的准确率。Li等人[14]提出一种基于Bi-LSTM神经网络模型的漏洞检测系统VulDeePecker。该系统将程序源码表示为语法、语义相关的代码小片段，通过Bi-LSTM神经网络模型实现了代码小片段粒度的漏洞检测。Zou等人[15,16]提出了两种漏洞检测系统μVulDeePecker和VulDeeLocator。这两种系统在VulDeePecker的基础上，分别实现了对多类型漏洞的检测和对漏洞位置的细粒度定位。Russell等人[17]通过卷积神经网络提取函数源码中的特征信息，并将提取的信息输入给后端分类器-随机森林以实现漏洞检测。Zhou等人[18]提出了一种基于图神经网络的漏洞检测系统Devign。该系统将程序源码表示为代码属性图，然后利用图神经网络学习代码属性图中的语义信息进行漏洞检测。段旭等人[19]提出了一种基于注意力机制的漏洞检测工具VulSniper。该工具将程序源码表示为代码属性图，并用144维向量来表示代码属性图中各节点之间的关系，通过注意力机制来捕捉关键的漏洞特征实现函数粒度的漏洞检测。

虽然上述不同结构的神经网络模型能从大量样本中学习出漏洞特征，但是研究人员难以从神经网络的黑盒模型中提取出分类特征对结果进行可解释性分析。通常用于分类的神经网络模型的输出结果为目标所属类别的概率值，即置信度，但并不清楚得出此结果的原因。当前基于深度学习的漏洞检测方法还难以生成PoC(Proof of Concept)来验证漏洞是否存在，而在后续深入分析漏洞成因时则需要具体的漏洞特征。如果能从神经网络模型中提取出漏洞的分类特征，则有助于研究人员对漏洞成因进行分析。目前，在图片分类中可运用神经网络可解释技术对分类结果的原因进行分析，如通过类激活映射(Class Activation Mapping, CAM)可以将图片中与目标类别相关的特征区域以热力图的形式进行展示[20]。Russell等人[17]在其论文中提到了可以通过CAM来定位更细粒度的漏洞位置，但他们并没有进行具体实现和实验验证。目前，还没有见到有其他文献将神经网络可解释性技术运用到漏洞检测当中。

当前针对基于深度学习的源代码漏洞检测存在以下困难和挑战。(1)程序源码信息的提取问题。类似于图片的像素值可以反映图片各部分特征，那么程序中的哪些信息可以反映漏洞特征？如何提取程序源码中与漏洞特征相关的信息是源码漏洞检测需要解决的首要问题。(2)程序源码信息的向量嵌入问题。神经网络模型以向量作为输入，因此从程序中提取的信息都要转换成向量，且经过词嵌入后的程序信息应当保留其语义相关性。由于源码中存在大量的自定义标识符，因此在词嵌入过程中会出现很多库外词。如何缓解库外词过多的问题，同时保留程序信息的语义相关性，也是源代码漏洞检测要解决的一个关键问题。(3)神经网络模型的可解释性。虽然神经网络模型能从数据集中学习到漏洞特征，从而实现对漏洞函数的判定，但是模型为什么做出这样的判定，判定的依据是什么？神经网络模型往往难以给出相应的解释。本文针对以上问题，提出一种基于卷积神经网络和全局平均池化的可解释性源代码漏洞检测方法。该方法以token序列作为程序信息的提取方式，对部分自定义标识符

进行归一化；然后采用One-hot编码方式生成词向量，构建以卷积神经网络和全局平均池化(Convolution Neural Networks and Global Average Pooling， CNN-GAP)为框架的神经网络进行漏洞检测；最后通过CAM方式对检测结果进行可解释分析。通过对比分析，本文提出的CNN-GAP模型的精确率、召回率等性能指标优于Russell等人提出的模型，模型具有较好的泛化性能，且能够实现漏洞函数代码行粒度的可解释性分析。

2 漏洞检测模型设计

本节详细阐述基于CNN-GAP可解释性模型漏洞检测方法的设计思路，主要包括函数源码的预处理和可解释性神经网络模型的构建方法。

2.1 函数源码预处理

函数源码预处理是将源代码中的函数转化为适合神经网络模型处理的数据。由于源代码中标识符的形式各异，为了缓解库外词过多的问题，需要将源代码中的自定义标识符进行归一化，用占位符代替原来的标识符。在对标识符进行归一化时，并不对所有标识符进行归一化，而是保留一些特殊的标识符，比如memcpy, strcpy, memset等内存操作和uint8_t, uint16_t等类型名。通过归一化，既可以缓解由于标识符命名多样化而造成语料库爆炸的问题，还可以保留程序中与内存操作和变量类型相关的语义信息。

归一化标识符即对函数名、变量名、字符串等进行归一化。首先通过正则表达式匹配出程序源代码中所有的函数名、变量名和字符串；然后用FUN+NUM替换函数名，用VAR+NUM替换变量名，用STRING+NUM替换字符串，最终得到归一化后的标识符。图1为对函数源码标识符进行归一化的示例，其中图1(a)为归一化前的示例函数源码，图1(b)为对图1(a)的源代码归一化后的函数源码形式。

```
1  Stack_Based_Buffer_Overflow()
2  {
3     char * data;
4     char dataBadBuffer[50];
5     char dataGoodBuffer[100];
6     goto source;
7   source:
8     data = dataBadBuffer;
9     data[0] = '\0';
10    {
11      char source[100];
12      memset(source, 'C', 100-1);
13      source[100-1] = '\0';
14      strcpy(data, source);
15      printLine(data);
16    }
17 }
```

(a) 源代码归一化前形式

```
1  Func1()
2  {
3     char * VAR4;
4     char VAR2[50];
5     char VAR1[100];
6     goto VAR3;
7   VAR3:
8     VAR4 = VAR2;
9     VAR4[0] = 'STRING1';
10    {
11      char VAR3[100];
12      memset(VAR3, 'STRING2', 100-1);
13      VAR3[100-1] = 'STRING1';
14      strcpy(VAR4, VAR3);
15      FUNC2(VAR4);
16    }
17 }
```

(b) 源代码归一化后形式

图 1 函数源码标识符归一化示例

函数源码作为特殊的文本序列，其分隔符不同于中英文文本序列。根据分隔符的不同样式，可建立3种分隔符的语料库，其中只含有1个字符的分隔符22个，只含有两个字符的分隔符21个，只含有3个字符的分隔符2个，具体分隔符分布如图2所示。

函数源码由一个个token组成。以上述分隔符为界，将每个函数切分为token序列，为了得到token的向量表示，用One-hot编码方式对token进行词向量嵌入。整个One-hot语料库包括了分隔符、归一化后的标识符、关键字等3种类型token。整个语料库共有255个词，其中3种类型的分隔符45个；归一化后的函数名20个、变量名20个、字符串10个，共50个标识符；C/C++中的关键字、内存操作库函数名、特殊变量名共160个。由于语料

```
operators3 = {'<<=', '>>='}
operators2 = {
   '->', '++', '--', '**', '!~',
   '<<', '>>', '<=', '>=', '==',
   '!=', '&&', '||', '+=', '-=',
   '*=', '/=', '%=', '&=', '^=', '|='
}
operators1 = {
   '(', ')', '[', ']', '{',
   '}', '.', '+', '&', ''',
   ''', '%', '<', '>', '^',
   '|', '=', ',', '?', ':',
   '!', '~'
}
```

图 2 3种形式的分隔符

库共255个词，因此每个token被编码成255维的词向量。为了得到固定形状大小的神经网络输入，将切分成token序列的函数源码进行截断或补齐。当token数量超出300个时，丢弃超出的部分，只保留前300个token。当token数量不足300个时，用255维的零向量补足300个。整个函数源码转换成形状为300×255的向量。

2.2 神经网络模型构建

神经网络模型既需要适配前端的输入形式，又需要满足后端的输出需求，即能够将输出的结果进行可解释性分析。为了能对输出结果进行可解释性分析，采用卷积神经网络和全局平均池化相结合的模型对输入向量进行处理，如图3所示。在对整个模型进行训练调参后，采用3层的卷积神经网络模型，其中卷积核的数量为128个，卷积核的大小分别为6, 5, 4，移动步长均为1。在前3层卷积计算中，为了使特征图中的激活值能与输入原图对应，将输出特征图补全到原图的形状大小。全局平均池化层将最后一个卷积层输出的每个特征图的激活值取平均，最后这些均值的加权和作为Softmax层的输入。

CNN-GAP模型中的卷积模块是1维卷积(Conv1D)，$f_k(x)$表示最后一个卷积层输出的第k(k=1, 2, …, 128)个特征图，空间坐标为x(x=1, 2, …, 300)处的激活值。对于第k个特征图，通过GAP后的结果为$F_k=\sum_x f_k(x)$。对于类别c(c=1, 2)，softmax层的输入为$S_c=\sum_k W_{c,k}F_k$，$W_{c,k}$代表第k个特征图对应类c的权重。实际上，$W_{c,k}$就是F_k对类c的重要性。最后类c的softmax输出为$P_c=\exp(S_c)/\sum c\exp(S_c)$。这里忽略偏差项bias，即将softmax的偏差项设置为0。将$F_k=\sum_x f_k(x)$代入S_c，得到

$$S_c=\sum_k W_{c,k}\sum_x f_k(x)=\sum_x\sum_k W_{c,k}f_k(x) \tag{1}$$

对于可解释的效果而言，不仅需要知道每个特征图对最后分类的重要性，还要知道特征图中每个位置x对最后分类的重要性。用$M_c(x)$($M_c(x)\in R$)表示特征图中每个空间位置x对类别c的重要性，输入softmax层的S_c还可以表示为

$$S_c=\sum_x M_c(x) \tag{2}$$

由式(1)和式(2)，可以得到

$$M_c(x)=\sum_k W_{c,k}f_k(x) \tag{3}$$

式(3)表明了特征图中空间位置x对划分为类别c的重要性$M_c(x)$，可以通过网络模型中的参数$f_k(x)$和$W_{c,k}$计算得出。CAM可解释表示形式$M_c(x)$就是不同空间区域的线性加权可视化。根据以上分析，可以通过计算每个token对分类的贡献度来实现漏洞的可解释性。由于token对分类的贡献度是根据神经网络模型计算得出的，因此神经网络模型对漏洞特征的拟合度决定了token贡献度表征的准确性。从理论上分析，基于卷积神经网络和全局平均池化的可解释性模型可以适用于不同类型的漏洞，但是漏洞可解释性的准确性受限于神经网络模型对漏洞检测的准确性。只有模型对漏洞检测的准确性高，漏洞可解释性的准确性才会高，而模型的准确性与数据集的大小紧密相关。当前，缓冲区溢出漏洞的数据集较之于其他类型漏洞的数据集要丰富很多，因此本文主要针对缓冲区溢出类型的漏洞对模型的性能进行实验测试分析。

3 实验测试分析

实验测试的硬件环境为20核Core(TM) i9-7900X 3.30 GHz的CPU、48 GB内存、1块Nvidia RTX 2080Ti GPU。软件环境以Ubuntu 18.04 LTS为操作系统，keras和tensorflow作为构建神经网络的前端和后端。

实验测试采用Russell论文中的数据集，该数据集是从Debian和Github上收集的漏洞函数集。这些函数集均来源于真实软件，包含了CWE-119,

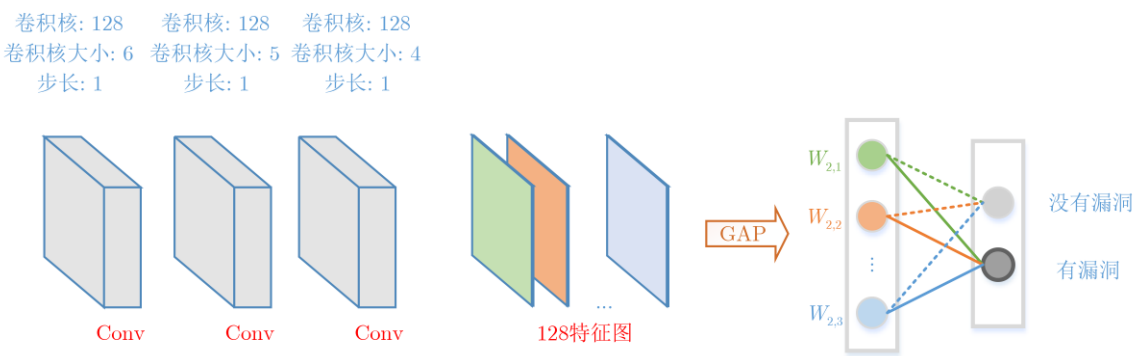

图 3 CNN-GAP模型结构

CWE-120, CWE-469和CWE-476等类型的漏洞函数，并由其团队成员对每个函数进行了确认和标记。由于CWE-119缓冲区溢出类型漏洞是目前影响最大的漏洞类型，且具有较大的数据量，因此本文在实验测试中采用CWE-119类型的漏洞函数作为基本数据集。将数据集中正负样本比例划分为1:3，即正样本为19286，负样本为57858；测试集中正负样本比例划分为1:1，即正样本为2452，负样本为2452。主要进行3项实验，包括2项模型对比实验以及1项输出结果可解释性分析实验。为了验证模型的有效性，实验1将CNN-GAP模型与 Russell模型在CWE-119类型的数据集上进行对比；实验2将已训练好的CNN-GAP模型与VulDeePecker模型，在VulDeePecker测试的CWE-119类型数据集上进行对比，以验证模型的泛化性能；实验3将CNN-GAP模型输出结果用CAM进行可视化输出，实现对漏洞特征的可解释性分析。

训练时将batch size设置为128，用binary_crossentropy作为损失函数，用Adam优化器进行优化。在Russell测试集上的结果如表1所示。从表1可以看出，模型的F1值达到了0.8872，Precision值达到了0.9531，Recall值达到了0.8299，模型的精确率较高但是检出率(True Positive Rate, TPR)较低，模型的误报率(False Positive Rate, FPR)不高但是存在一定的漏报率(False Negative Rate, FNR)。由于实验数据集来自Russell论文公开的数据集，因此将两个模型的实验结果进行对比，以说明CNN-GAP模型的有效性。

图4为CNN-GAP模型和Russell模型在CWE-119类型漏洞函数进行实验测试的 Precision-Recall对比图。对于Russell模型，当Recall值小于0.8时候，Precision值下降速度较缓慢；当 Recall值大于0.8时，Precision值迅速下降。对于CNN-GAP模型，其变化趋势与Russell模型相同，随着Recall值的增加，Precision值逐渐降低。当Recall值小于0.8时，Precision值下降十分缓慢，且均在0.9以上；当Recall值大于0.8时，Precision值虽然下降速度也很快，但是都在0.5以上，而在Russell模型中，当其Precision值超过0.8时，Recall值小于0.5；当Recall值大于0.5时，Precision值已经低于0.5了。从这些数据可以说明，Russell模型在保持较高的检测精确率时，召回率低即漏报率高；当保持较高召回率时，其检测的准确率又低于0.5。CNN-GAP模型不仅能够保持较高精确率，其召回率也不低。由于Russell模型对token进行归一化的方式是将token替换成字符串、浮点数等类型，然而常量、变量、函数名等有可能是相同的数据类型，因此它们有可能被替换成同一种占位符，使得标识符之间容易产生混淆。然而，CNN-GAP模型对每个token都采用不同的占位符来表示，不会造成替换后的标识符出现混淆的情况。因此，CNN-GAP 模型对CWE-119类型的漏洞检测效果优于Russell模型。

当正样本和负样本的数量不均衡时，还需要TPR和FPR来对实验结果进行评估。此时，根据阈值的取值不同，可以得到不同的FPR和TPR值。将所有的值以FPR为横轴、TPR为纵轴画图，得到ROC曲线图，曲线向下覆盖的面积即为AUC值。理想情况下AUC值为1.0，此时该分类器为完美分类器。一般情况下，AUC值在0.5～1.0，优于随机预测。图5为CNN-GAP模型采用Russell测试集得到的ROC曲线，AUC值为0.95，接近1.0，说明模型整体性能较好。

VulDeePecker是基于Bi-LSTM的源代码漏洞检测模型，它将程序源代码通过定义的规则切分成

表 1 CNN-GAP模型在Russell测试集上的结果

	Accuracy	Precision	Recall	F1	TPR	FPR	FNR
CNN-GAP	0.8945	0.9531	0.8299	0.8872	0.8299	0.0407	0.1700

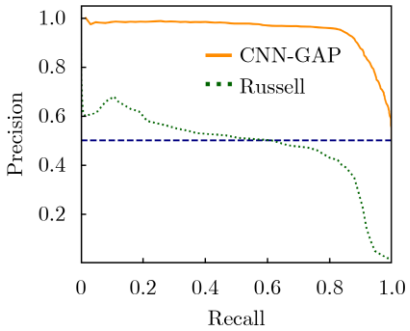

图 4 CNN-GAP和Russell模型的Precision-Recall曲线图

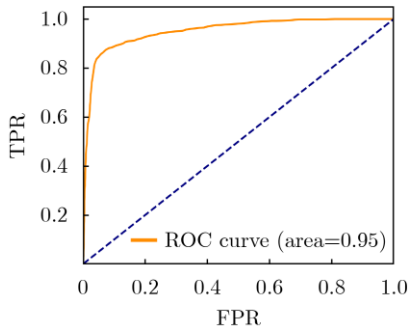

图 5 CNN-GAP模型的ROC曲线图

code gadget，然后在code gadget粒度上进行检测。该模型采用的数据集包括两种类型的漏洞，CWE-119和CWE-399。本实验主要针对CWE-119类型的数据集进行测试。VulDeePecker的CWE-119数据集是txt文档，以“–”作为分割符，有10440个含有漏洞的code gadget，29313个不含漏洞的code gadget。为了提高测试的效率和真实性，按照其数据集原本的数据排列顺序，取出前1000个不含漏洞的code gadget和最后1000个含漏洞的 code gadget进行测试，测试结果如表2所示。

从表2可以看出，CNN-GAP模型检测的精确率precision高于VulDeePecker模型，Recall，F1等参数略低于VulDeePecker模型。然而，CNN-GAP模型并没有在VulDeePecker模型的数据集上进行训练，仅仅在其测试集上进行实验，而实验结果非常接近VulDeePecker模型，由此可以说明CNN-GAP模型的泛化性能较好。在前期对程序源码的预处理中，VulDeePecker模型需要将程序进行分析切片提取，其前向切片和后向切片只保留了其关注的程序中几种易出现漏洞的元素(如API调用、数组变量、指针变量等)及其控制流和数据流依赖，丢弃了其他不属于此范围的代码元素。因此，VulDeePecker模型虽然精简了代码量，但是也丢失了部分代码特征。CNN-GAP模型无需对程序进行切片，直接对整个函数源码进行处理，不仅简化了预处理过程，而且包含了整个函数中所有元素的控制流和数据流依赖。从此角度考虑，CNN-GAP模型的实用性优于VulDeePecker模型。

通常情况下，分类模型的输出是判别待测目标为某个类别的概率值，可以依据这个概率值对待测目标进行分类。对于使用softmax作为判决器的二分类任务，当某个类别的输出值较大时，表明被测目标为该类别。由于模型的输出值是根据特征图中各位置的激活值得到的，因此通过对GAP层输出信息进行可解释性分析，可以计算得到源码中每个token对分类结果的重要性。为便于将每一个激活值与每一个token对应，在构建神经网络模型时，每层网络输出的特征图大小保持不变，因此激活值与神经网络权重系数的乘积可以表示token对某个类别的贡献度，且该数值越大，贡献度越大。若该数值为正，表明token对整个函数含有漏洞为正向激励；若该数值为负，则为反向激励。在源码预处理中，函数源码被切分为300个token序列。如果将每个token对最后分类结果的重要性进行表示，那么300个token在分布图中十分拥挤，不便于展示和理解。为了更好地从视觉上感受到函数源码对目标类别的重要性，实验3将GAP中的特征映射到函数源码中的每条语句，而不是每个token。在上文的输出结果可视化计算中已经详细说明了token权重的计算方法，最后将程序语句中每个token的重要性(权重值)进行相加作为此程序语句对分类结果的重要性。

下面的两段代码为VulDeePecker测试集中的函数，分别对应的是堆溢出漏洞和栈溢出漏洞。

图6的代码片段是CWE-119类型的堆溢出漏洞，其中memset函数的功能是复制字符“C”到source所指区域的前100-1个字符，strncat函数的功能是把source所指向的字符串追加到data所指向的字符串的结尾，直到100个字符长度。因为data是动态分配的堆内存，而source所指字符串的内存大小可能会超出data的内存大小，则会造成堆溢出。由于strncat函数没有对source的大小进行检查，因此存在堆溢出漏洞。图7的代码片段是CWE-119类型的栈溢出漏洞，其中memset函数的功能是复制字符“A”到data所指区域的前100-1个字符，memcpy函数的功能是把data内存中的字符串复制到dest内存区域中。dest是数组，为栈内存区域。然而，在进行复制操作时没有对dest的内存大小和复制字符串的大小进行比较，因此复制的字符串长度超出了dest的内存大小，会造成栈溢出。

图8、图9分别为上述两段代码经过神经网络模型判别后，根据可解释性分析输出的可视化结果，其中数值大于0表示对应的代码行与漏洞相关。可以看出图8中，行10—行13与漏洞特征相关，这几行代码包含了strncat漏洞代码、data堆内存、source[]数组，直观上能表示出漏洞特征对最后分类的重要性。图9中表明第12，13行与漏洞特征相关，这几行代码包含了memcpy漏洞代码，直观上能表示出漏洞特征对最后分类的重要性。因此，从可视化的柱状图中可以看出CNN-GAP模型所学到的“漏洞特征”在程序源码中的分布情况。换个角度，此漏洞特征也可以表明漏洞代码在整个源码中

表 2　CNN-GAP模型与VulDeePecker模型的实验结果对比

	Accuracy	Precision	Recall	F1	TPR	FPR	FNR
CNN-GAP	0.8490	0.9235	0.7610	0.8344	0.7610	0.0630	0.2390
VulDeePecker	–	0.9170	0.8200	0.8660	0.8200	0.0290	0.1800

```
1  Heap_Overflow()
2  {
3     char * data;
4     data = NULL;
5     if(staticTrue)
6     {
7         data = new char[50];
8         data[0] = '\0';
9     }
10    char source[100];
11    memset(source, 'C', 100-1);
12    source[100-1] = '\0';
13    strncat(data, source, 100);
14    printLine(data);
15    delete [] data;
16 }
```

图 6 堆溢出示例代码

```
1  Stack_Overflow()
2  {
3     char * data;
4     char * &dataRef = data;
5     char dataBuffer[100];
6     data = dataBuffer;
7     memset(data, 'A', 100-1);
8     data[100-1] = '\0';
9     {
10        char * data = dataRef;
11        {
12            char dest[50] = '';
13            memcpy(dest, data,
                 strlen(data)*sizeof(char));
14            dest[50-1] = '\0';
15            printLine(data);
16        }
17    }
18 }
```

图 7 栈溢出示例代码

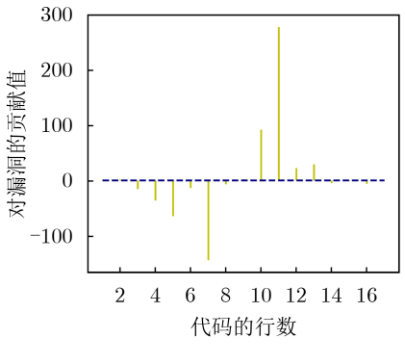

图 8 CNN-GAP模型针对堆溢出示例代码的可视化结果

的位置，即从柱状图中可以得知函数源码中漏洞的大概位置，实现在代码行粒度的漏洞定位。

4 结束语

本文以CNN-GAP可解释性模型为神经网络框架对CWE-119缓冲区溢出类型的漏洞函数进行检测，并通过CAM可解释技术实现了结果的可视化输出。通过与Russell提出的神经网络模型及VulDeePecker模型进行对比分析，验证了CNN-GAP模型的有效性和泛化性。当然，本文提出的模型还可以在以下3个方面进一步完善。一是在程序建模中，虽然通过对自定义标识符进行归一化处理来缓解库外词过多的问题，但同时也丢失了其中的部分语义信息。在后续的工作中可以考虑采用更为完善的处理方式以减少语义信息的损失。二是在神经网络可解释性分析中，只是大致将程序源码中与漏洞相关的代码行进行了高亮表示，没有对其根本原因进行分析。在后续的工作中力争实现更为细粒度的漏洞函数定位分析，达到token粒度的可视化输出。三是当前的模型仅能识别CWE-119缓冲区溢出类型漏洞，后续可利用CWE-120，CWE-469等类型的漏洞函数集对模型进行训练，使模型能实现多种类型漏洞的检测。

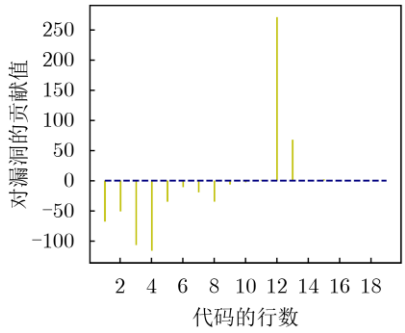

图 9 CNN-GAP模型针对栈溢出示例代码的可视化结果

参 考 文 献

[1] COUSOT P, COUSOT R, FERET J, *et al*. The ASTRÉE analyzer[C]. The 14th European Symposium on Programming, Edinburgh, UK, 2005: 21–30.

[2] HOLZMANN G J. The model checker SPIN[J]. *IEEE Transactions on Software Engineering*, 1997, 23(5): 279–295. doi: 10.1109/32.588521.

[3] YAMAGUCHI F, LOTTMANN M, and RIECK K. Generalized vulnerability extrapolation using abstract syntax trees[C]. The 28th Annual Computer Security Applications Conference, New York, USA, 2012: 359–368.

[4] YAMAGUCHI F, GOLDE N, ARP D, *et al*. Modeling and discovering vulnerabilities with code property graphs[C]. The 2014 IEEE Symposium on Security and Privacy, Berkeley, USA, 2014: 590–604.

[5] MILLER B P, FREDRIKSEN L, and SO B. An empirical study of the reliability of UNIX utilities[J]. *Communications of the ACM*, 1990, 33(12): 32–44. doi: 10.1145/96267.96279.

[6] STEPHENS N, GROSEN J, SALLS C, *et al*. Driller: Augmenting fuzzing through selective symbolic execution[C]. The 2016 23rd Network and Distributed System Security Symposium, San Diego, USA, 2016: 1–16.

[7] PORTOKALIDIS G, SLOWINSKA A, and BOS H. Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation[C]. The 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, New York, USA, 2006: 15–27.

[8] 邹权臣, 张涛, 吴润浦, 等. 从自动化到智能化: 软件漏洞挖掘技术进展[J]. 清华大学学报:自然科学版, 2018, 58(12): 1079–1094.
ZOU Quanchen, ZHANG Tao, WU Runpu, *et al*. From automation to intelligence: Survey of research on vulnerability discovery techniques[J]. *Journal of Tsinghua University:Science & Technology*, 2018, 58(12): 1079–1094.

[9] LIN Guanjun, ZHANG Jun, LUO Wei, *et al*. POSTER: Vulnerability discovery with function representation learning from unlabeled projects[C]. The 2017 ACM SIGSAC Conference on Computer and Communications Security, New York, USA, 2017: 2539–2541.

[10] LIN Guanjun, ZHANG Jun, LUO Wei, *et al*. Cross-project transfer representation learning for vulnerable function discovery[J]. *IEEE Transactions on Industrial Informatics*, 2018, 14(7): 3289–3297. doi: 10.1109/TII.2018.2821768.

[11] LIN Guanjun, ZHANG Jun, LUO Wei, *et al*. Software vulnerability discovery via learning multi-domain knowledge bases[J]. *IEEE Transactions on Dependable and Secure Computing*, 2021, 18(5): 2469–2485. doi: 10.1109/TDSC.2019.2954088.

[12] LIU Shigang, LIN Guanjun, HAN Qinglomg, *et al*. DeepBalance: Deep-learning and fuzzy oversampling for vulnerability detection[J]. *IEEE Transactions on Fuzzy Systems*, 2020, 28(7): 1329–1343.

[13] LIU Shigang, LIN Guanjun, QU Lizhen, *et al*. CD-VulD: Cross-domain vulnerability discovery based on deep domain adaptation[J]. *IEEE Transactions on Dependable and Secure Computing*, 2022, 19(1): 438–451. doi: 10.1109/TDSC.2020.2984505.

[14] LI Zhen, ZOU Deqing, XU Shouhuai, *et al*. VulDeePecker: A deep learning-based system for vulnerability detection[C]. The 25th Annual Network and Distributed System Security Symposium, San Diego, USA, 2018.

[15] ZOU Deqing, WANG Sujuan, XU Shouhuai, *et al*. μVulDeePecker: A deep learning-based system for multiclass vulnerability detection[J]. *IEEE Transactions on Dependable and Secure Computing*, 2021, 18(5): 2224–2236. doi: 10.1109/TDSC.2019.2942930.

[16] LI Zhen, ZOU Deqing, XU Shouhuai, *et al*. VulDeeLocator: A deep learning-based fine-grained vulnerability detector[J]. *IEEE Transactions on Dependable and Secure Computing*, 2022, 19(4): 2821–2837. doi: 10.1109/TDSC.2021.3076142.

[17] RUSSELL R, KIM L, HAMILTON L, *et al*. Automated vulnerability detection in source code using deep representation learning[C]. The 17th IEEE International Conference on Machine Learning and Applications, Orlando, USA, 2018: 757–762.

[18] ZHOU Yaqin, LIU Shangqing, SIOW J K, *et al*. *Devign*: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]. The 33rd International Conference on Neural Information Processing Systems, Red Hook, USA, 2019: 10197–10207.

[19] 段旭, 吴敬征, 罗天悦, 等. 基于代码属性图及注意力双向LSTM的漏洞挖掘方法[J]. 软件学报, 2020, 31(11): 3404–3420.
DUAN Xu, WU Jingzheng, LUO Tianyue, *et al*. Vulnerability mining method based on code property graph and attention BiLSTM[J]. *Journal of Software*, 2020, 31(11): 3404–3420.

[20] ZHOU Bolei, KHOSLA A, LAPEDRIZA A, *et al*. Learning deep features for discriminative localization[C]. 2016 IEEE conference on Computer Vision and Pattern Recognition, Las Vegas, USA, 2016: 2921–2929.

王　剑：男，1975年生，教授，博士生导师，研究方向为漏洞挖掘与分析、网络安全协议设计与分析、网络安全测试与评估.

匡洪宇：男，1996年生，博士生，研究方向为漏洞挖掘与分析.

李瑞林：男，1982年生，副教授，研究方向为漏洞挖掘与分析、网络安全协议设计与分析.

苏云飞：男，1982年生，博士，研究方向为漏洞挖掘与分析、网络渗透测试分析.

责任编辑：马秀强