计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190800046

基于 BiLSTM 模型的漏洞检测

龚扣林　周　宇　丁　笠　王永超
南京航空航天大学计算机科学与技术学院　南京 211100
高安全系统的软件开发与验证技术工信部重点实验室　南京 211100
(506531906@qq.com)

摘　要　随着计算机技术应用的不断深化,软件的数量和需求不断增加,开发难度不断升级。代码复用以及代码本身的复杂度,使得软件中不可避免地引入了大量漏洞。这些漏洞隐藏在海量代码中很难被发现,但一旦被人利用,将导致不可挽回的经济损失。为了及时发现软件漏洞,首先从源代码中提取方法体,形成方法集;为方法集中的每个方法构建抽象语法树,借助抽象语法树抽取方法中的语句,形成语句集;替换语句集中程序员自定义的变量名、方法名及字符串,并为每条语句分配一个独立的节点编号,形成节点集。其次,运用数据流和控制流分析提取节点间的数据依赖和控制依赖关系。然后,将从方法体中提取的节点集、节点间的数据依赖关系以及控制依赖关系组合成方法对应的特征表示,并运用 one-hot 编码进一步将其处理为特征矩阵。最后,为每个矩阵贴上是否含有漏洞的标签以生成训练样本,并利用神经网络训练出相应的漏洞分类模型。为了更好地学习序列的上下文信息,选取了双向长短时记忆网络(Bidirectional Long Short-Term Memory Networks,BiLSTM)神经网络,并在其上增加了 Attention 层,以进一步提升模型性能。实验中,漏洞检测结果的精确率和召回率分别达到了 95.3%和 93.5%,证实了所提方法能够较为准确地检测到代码中的安全漏洞。

关键词:漏洞检测;特征表示;BiLSTM;Attention;分类模型
中图法分类号　TP305

Vulnerability Detection Using Bidirectional Long Short-term Memory Networks

GONG Kou-lin,ZHOU Yu,DING Li and WANG Yong-chao
School of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211100,China
Ministry Key Laboratory for Safety-critical Software Development and Verification,Nanjing 211100,China

Abstract　With the continuous development of the application of computer technology,the number and demand of software continue to increase,and the difficulty of development is constantly escalating. Code reuse and the complexity of the code itself have inevitably introduced a number of vulnerabilities in software. These vulnerabilities hidden in massive code are hard to find. But once they are exploited by people,it will lead to irreparable economic losses. In order to discover software vulnerabilities in time,firstly,this paper extracts the method body from the source code to form a method set,and then constructs an abstract syntax tree for each method in the method set. The statements in the method are extracted by means of the abstract syntax tree to form a statement set. The customized variable name,method name and string with some uniform identifiers are replaced. A separate node number is assigned to each statement to form a node set. Secondly,data flow and control flow analysis are used to extract data dependencies and control dependencies between nodes. Then,the node set extracted from the method body,the inter-node data dependency relationship and control dependency relationship are combined into a feature representation corresponding to the method,and further processed into a feature matrix by using one-hot encoding. Finally,each matrix is labeled with a vulnerability tag to generate training samples,and a neural network is used to train the corresponding vulnerability classification model. In order to learn the context information of the sequence better,the BiLSTM network is selected and the Attention layer is added to further improve the performance of the model. In the experiment,the accuracy and recall rate of the vulnerability detection results reach 95.3% and 93.5% respectively,which confirmes that the proposed method can detect the security vulnerabilities in the code more accurately.

Keywords　Vulnerability detection,Feature representation,BiLSTM,Attention,Classification model

到稿日期:2019-08-09　返修日期:2019-11-27　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61972197);中央高校基本科研业务项目(NS2019055)
This work was supported by the National Natural Science Foundation of China(61972197)and Fundamental Research Funds for the Central Universities(NS2019055).
通信作者:周宇(zhouyu@nuaa.edu.cn)

1 引言

软件漏洞是一个软件中的特定缺陷或疏忽，允许攻击者执行恶意行为，包括暴露敏感信息、控制程序运行和破坏计算机系统等。随着计算机技术应用的不断深化，软件的需求和规模不断增加，代码的数量和复杂度呈指数级增长。一方面，开发人员的水平参差不齐，庞大的代码量给开发人员的测试与维护带来了极大的挑战；另一方面，为了提高软件开发效率，降低开发成本，代码复用得到了推崇。由此，软件中暴露出的安全漏洞也与日俱增，对公司和个人都造成了重大损害[1]。例如，美国计算机应急准备小组(United States Computer Emergency Readiness Team，US-CERT)[2]披露，流行浏览器插件(Adobe Flash Player，Oracle Java 等)中的漏洞直接威胁到数百万互联网用户的安全和隐私。同时，一些基础和流行的开源软件(Heartbleed，ShellShock 等)中的漏洞也威胁到全球数千家公司及其客户的安全，造成的直接或间接经济损失不可估量。因此，快速且准确地检测代码中的漏洞已成为软件行业和计算机安全领域的一个至关重要的课题。

由于应用程序会接收各种各样的未知输入，并且需要通过大规模的复杂代码实现，因此在大量代码中搜寻少量漏洞无异于“大海捞针”[3]。传统的漏洞检测方法多依赖于领域专家制定相应的规则，这不仅需要消耗大量的人力资源，而且由于制定者的水平参差不齐，检测结果的准确性难以得到保障。

随着机器学习的不断兴起和成熟，其强大的学习能力受到了人们的关注。为了减轻对人工特征提取的依赖性，研究人员开始尝试将机器学习引入漏洞检测领域以自动学习漏洞特征。为了充分学习样本序列的上下文信息，本文选取了双向长短时记忆网络。BiLSTM 是对 LSTM 神经网络的扩展，通过加入一层输入序列的反转副本为网络提供额外的上下文，继而提升模型在序列分类问题上的表现。此外，为了进一步提高结果的准确率，在 LSTM 层和输出层之间又增加了 Attention 层，以调整各时序所占权重。

本文工作的主要贡献有以下两个方面：

1)结合抽象语法树分析、数据流分析和控制流分析技术，提出了一种融合代码结构、数据依赖和控制依赖信息的源代码特征提取方法；

2)结合深度神经网络技术和程序静态分析技术，提出了一种基于 BiLSTM 模型的漏洞检测方法。

本文第 2 节介绍相关工作；第 3 节阐述本文提出的漏洞检测方法；第 4 节展示实验设置并对实验结果进行分析；第 5 节讨论所提方法的有效性威胁；最后总结全文。

2 相关工作

传统的漏洞检测方法可以分为以下几类：1)探索输入空间，如黑盒模糊测试[4]和灰盒模糊测试[5-8]，此类方法效率较低，且无法产生输入来检查程序中的所有状态；2)分析程序的内部状态空间，如符号执行[9-10]，此类方法受路径探索问题和 SMT 求解器能力的限制，可扩展性较差；3)基于领域专家制定的模式、规则或指标分析程序[11-16]，此类方法需要耗费大量的人力，且查准率和查全率得不到保证。

近年来，机器学习的不断发展也催生了许多与深度学习技术相结合的工作[17-22]。基于深度学习的方法免去了手动定义特征的过程，可以自动学习易受攻击代码的模式。Scandariato 等[17]选取了 3 个大型的 PHP 开源项目(Drupal，Moodle，PhpMyAdmin)作为数据集，并分别利用度量元特征和文本挖掘两种方法检测漏洞，结果显示文本挖掘的方法明显优于度量元特征。Yamaguchi 等[18]提出了一种名为 vulnerability extrapolation 的方法，该方法首先为每个函数构建抽象语法树，然后利用词袋技术将提取的抽象语法树映射到向量空间，最后运用潜在语义分析，先将所有函数的向量表示组合成一个大的稀疏矩阵，继而运用矩阵奇异值分解来识别在语法树中频繁出现的结构化模式。Russell 等[19]提出了一个大规模的函数级漏洞检测系统，用于在语法分析后学习源代码的深度特征表示。该系统将函数源代码的特征表示与随机森林相结合，以生成分类器。Harer 等[20]使用机器学习方法执行数据驱动的漏洞检测，并对使用源代码作为训练集和使用编译代码作为训练集的两种方式的有效性进行比较。上述方法的粒度是函数级别，无法确定漏洞的精确位置。VulDeePecker[21]首次使用深度学习来检测切片级别(比函数级别更精细)的漏洞，可以做到漏洞定位。SySeVR[22]是一个基于深度学习的框架，它使用基于语法、基于语义和向量的表示来检测切片级别的各种漏洞，但是没有系统地比较不同因素对漏洞检测有效性的定量影响。

3 方法介绍

3.1 整体架构

本文提出的基于 BiLSTM 模型的漏洞检测方法的整体架构如图 1 所示。

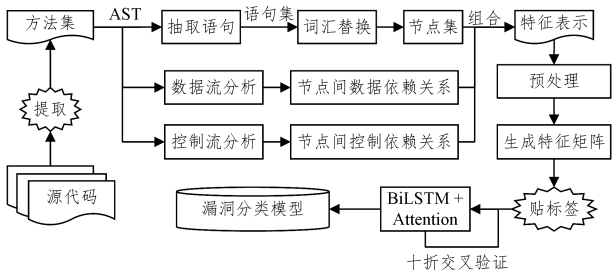

图 1 方法整体架构图

Fig. 1 Overall architecture of the proposed method

首先，从源代码中提取方法体，形成方法集；为方法集中的每个方法构建抽象语法树，借助抽象语法树抽取方法中的语句，形成语句集；替换语句集中程序员自定义的变量名、方法名及字符串，并为每条语句分配一个独立的节点编号，形成节点集。其次，运用数据流和控制流分析提取节点间的数据依赖和控制依赖关系。然后，将从方法体中提取的节点集、节点间数据依赖关系以及控制依赖关系组合成方法对应的特征表示，并运用 one-hot 编码进一步将其处理为特征矩阵。最后，为每个矩阵贴上是否含有漏洞的标签以生成训练样本，并增设 Attention 机制来辅助 BiLSTM 神经网络训练出更好的漏洞分类模型。

3.2 节点集

3.2.1 抽象语法树

生成节点集，首先需要为方法生成抽象语法树。该功能

的实现基于开源工具 ANTLR4[23]。以图 2 所示的一段含有缓冲区溢出漏洞(CWE[24]-120：Buffer Copy without Checking Size of Input)的经典代码为例，其生成的抽象语法树可扫描 OSID 码查看。

```
…
  struct hostent *clienthp;
  char hostname[MAX_LEN];
  //create server socket, bind to server address and listen on socket
  …
  //accept client connections and process requests
  int count=0;
  for (count=0;count<MAX_CONNECTIONS;count++) {
    int clientlen=sizeof(struct sockaddr_in);
    int clientsocket=accept(serversocket,(struct sockaddr *)&clientaddr,
    &clientlen);
    if (clientsocket >= 0) {
      clienthp=gethostbyaddr((char *) &clientaddr.sin_addr.s_addr,si-
      zeof(clientaddr.sin_addr.s_addr),AF_INET);
      strcpy(hostname,clienthp->h_name);
      logOutput("Accepted client connection from host",hostname);
      //process client request
      …
      close(clientsocket);
    }
  }
  close(serversocket);
…
```

图 2 缓冲区溢出漏洞示例

Fig. 2 An example of buffer overflow vulnerability

在该示例中，服务器接受来自客户端的连接并处理客户端请求。接受客户端连接后，程序将使用 gethostbyaddr 方法获取客户端信息，复制链接到本地变量的客户端的主机名，并将客户端的主机名输出到日志文件。但是，客户端的主机名可能会超出为本地主机名所对应变量分配的大小，因此，当使用 strcpy 方法将客户端主机名复制到本地变量时，将导致缓冲区溢出。

3.2.2 生成语句集

基于生成的抽象语法树提取方法体中的语句，形成如图 3 所示的语句集，该集合不包含方法原有的控制逻辑。

```
1. struct hostent *clienthp
2. char hostname[MAX_LEN]
3. int count=0
4. count=0
5. count<MAX_CONNECTIONS
6. count++
7. int clientlen=sizeof(struct sockaddr_in)
8. int clientsocket=accept(serversocket,(struct sockaddr *)&clientaddr,
   &clientlen)
9. clientsocket >= 0
10. clienthp=gethostbyaddr((char *) &clientaddr.sin_addr.s_addr,sizeof
    (clientaddr.sin_addr.s_addr),AF_INET)
11. strcpy(hostname,clienthp->h_name)
12. logOutput("Accepted client connection from host",hostname)
13. close(clientsocket)
14. close(serversocket)
```

图 3 语句集

Fig. 3 Set of statements

3.2.3 生成节点集

根据抽象语法树中的节点类型，筛选出用户自定义的变量名、方法名以及字符串，并对其做统一替换，以在将文本表示转换为向量时压缩词汇表的大小。例如，同样是 bool 类型的变量，程序员 A 命名为 *isTrue*，程序员 B 命名为 *isRight*，程序员 C 命名为 *exist* 等，上万个程序员可能产生数千个变量名，将这些自定义的变量名统一替换为某一标识符(如 cid)之后，可以有效地减少词汇的数量，从而在将 token 展开为 one-hot 向量时降低向量的维度。同一方法体中的不同变量用 *cid*_1 和 *cid*_2 进行区分。最后，将每一个处理完的语句当作方法的一个节点，并为其分配一个独立的节点编号，形成如图 4 所示的节点集。

```
n1:struct hostent * cid_1
n2:char cid_2 [cid_3]
n3:int cid_4=0
n4:cid_4=0
n5:cid_4<cid_5
n6:cid_4++
n7:int cid_6=sizeof(structsockaddr_in)
n8:int cid_7=cid_8 (cid_9,(structsockaddr *) & cid_11,& cid_6 )
n9:cid_7>=0
n10:cid_1=cid_10((char *) & cid_11.cid_12.cid_13,sizeof(cid_11.cid_
    12.cid_13),cid_14)
n11:cid_15(cid_2,cid_1->cid_16)
n12:cid_17("cstring_1",cid_2)
n13:cid_18(cid_7)
n14:cid_18(cid_9)
```

图 4 节点集

Fig. 4 Set of nodes

3.3 节点间的数据依赖

运用数据流分析技术提取各节点对应语句之间的数据依赖关系，并保存为图 5 所示样式。其中，箭头表示数据的流向，如第 1 行中的 $n2 \rightarrow n11$ 即表示 $n11$ 节点语句的运行受 $n2$ 节点运行结果的影响，label 属性则表明产生影响的单元为 *cid*_2(hostname)变量。

$n2 \rightarrow n11$	label:*cid*_2
$n2 \rightarrow n12$	label:*cid*_2
$n4 \rightarrow n5$	label:*cid*_4
$n4 \rightarrow n6$	label:*cid*_4
$n6 \rightarrow n5$	label:*cid*_4
$n6 \rightarrow n6$	label:*cid*_4
$n8 \rightarrow n9$	label:*cid*_7
$n8 \rightarrow n13$	label:*cid*_7
$n10 \rightarrow n11$	label:*cid*_1

图 5 节点间的数据依赖关系

Fig. 5 Data dependencies between nodes

3.4 节点间的控制依赖

运用控制流分析技术提取语句之间的控制依赖关系，并保存为图 6 所示样式。如第 1 行和第 2 行所示，$n5$ 节点是一个比较表达式，该节点的运行会产生 True 和 False 两种结果。当结果为 True 时，跳转至 $n7$ 节点运行；反之则跳转至 $n14$ 节点。

$n5 \rightarrow n7$	label:True
$n5 \rightarrow n14$	label:False
$n9 \rightarrow n6$	label:False
$n9 \rightarrow n10$	label:True

图 6 节点间的控制依赖关系

Fig. 6 Control dependencies between nodes

3.5 特征矩阵

3.5.1 特征表示

以分号为间隔，将节点集、节点间的数据依赖关系和节点间的控制依赖关系中所有语句组合而成的文本，即为方法体所对应的特征表示。

3.5.2 预处理

以空格为分隔符对文本做分词处理，即可将每个方法对应的特征表示转换为一个 token 序列。为了使数据可训练，要求每个方法体对应的向量是固定维度，因此要求方法转换而成的 token 序列是固定长度。根据 token 序列长度分布的统计情况(如图 7 所示)，筛除长度超过 400 的 token 序列(长度超过 400 的约占总数量的 1.77%，舍弃这极小部分样本对实验结果的影响可忽略不计)；此外，长度小于 10 的 token 序列由于信息量过少，也一并筛除。

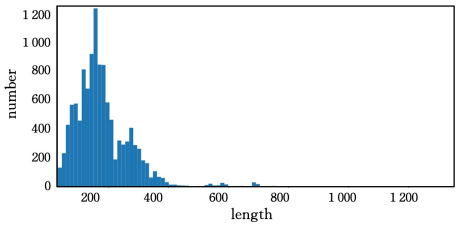

图 7 token 序列长度分布直方图

Fig. 7 Histogram of token sequences' length

将筛选后的 token 序列中出现的 token 不重复地加入到词汇表中，并依次为词汇表中的每个单词分配一个唯一且连续的正整数作为索引。将 token 序列中的每个 token 替换为其对应的索引值，由此每个方法即转换为一个长度不超过 400 的整数序列。以 0 填充长度不足 400 的整数序列，直至所有序列长度均为 400。

3.5.3 生成特征矩阵

使用 one-hot 编码将代码 token 映射到向量空间。代码 token 作为一种离散型特征，使用 one-hot 编码计算特征之间的距离会更加合理：1)解决了分类器不好处理离散数据的问题；2)在一定程度上起到了扩充特征的作用；3)解决了 token 长度不一致的问题。当然，该编码的缺点也很明显：1)它是一个词袋模型，没有考虑词与词之间的顺序信息，该缺点可以通过选取 BiLSTM 神经网络学习序列信息得到有效解决；2)该编码生成的特征过于稀疏，因此需要在模型中引入 Embedding 层对向量进行压缩。

将每个整数按照词汇表的长度展开，且将对应的索引位置 1，其余位置 0，即可得到该整数所对应的 token 的 one-hot 向量。由此，所有的方法均已转换为长度为 len(词汇表) * 400 的特征矩阵。

3.6 模型训练

按照方法是否含有漏洞为该方法对应的特征矩阵添加相应的标签：有漏洞为 1，没有漏洞为 0。利用基于 Attention 的双向长短时记忆神经网络在含有标签的数据集上训练出分类模型。基于 Attention 的 BiLSTM 神经网络的架构如图 8 所示。

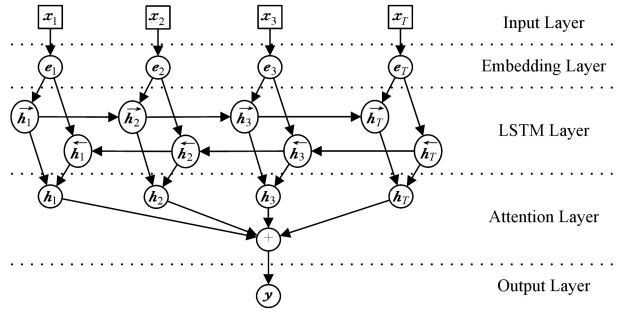

图 8 基于 Attention 的 BiLSTM

Fig. 8 BiLSTM based on attention

3.6.1 Embedding

训练数据样本的不断增多，词汇表的长度不断增加，导致方法对应的特征矩阵中向量的维度过大。冗长的向量不仅信息过于稀疏，不利于机器学习到有用信息；而且会消耗大量的计算资源，给计算机的软硬件带来极大的负担。因此，在输入层导入数据之后，需要引入 embedding 层对原始向量进行压缩。Embedding 的原理如图 9 所示。

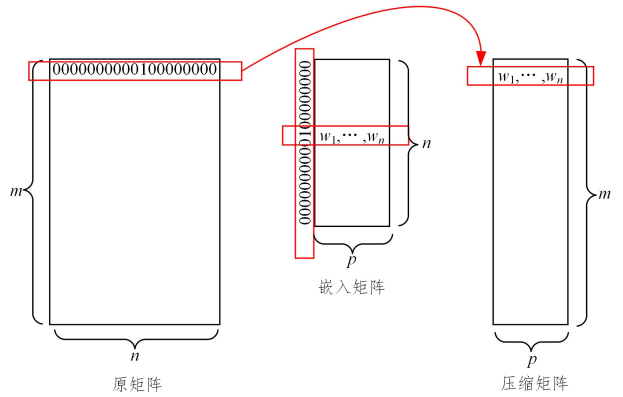

图 9 Embedding 的原理

Fig. 9 Principle of Embedding

3.6.2 BiLSTM

BiLSTM，双向长短时记忆网络，顾名思义，由前向 LSTM 和后向 LSTM 组合而成。LSTM 作为 RNN 的变种，很好地解决了传统 RNN 由于梯度消失现象难以处理长序列数据的问题。LSTM 由于能够学习获得数据的序列信息，在学术界和工业界受到了极大追捧。双向 LSTM 因为组合了序列的正反双重信息，能够获得更好的学习效果，所以是更优的选择。

3.6.3 Attention

Attention 模拟的是人的注意力模型，人从外界接收信息时，对各种信息的取舍并不是均衡的，而是根据需求来调整相应的权重，以合理地进行资源分配。同样，对于海量的代码信

息，也应该按照一定的权重来取舍，因此需要在神经网络中引入 Attention 层。不增设 Attention 层时，BiLSTM 会使用最后一个时序的输出向量作为特征向量，然后进行 softmax 分类。增设了 Attention 层以后，该层会先计算每个时序的权重，然后将所有时序的向量进行加权求和作为特征向量，以得到分类结果。

4 实验验证

为了验证上述方法的有效性，将其与现有的 2 项较新的基于机器学习的工作进行对比实验。实验所用数据集是在美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）的软件保障参考数据集（Software Assurance Reference Dataset，SARD）[25] 上选取的 3 个子集（3 个子集在大的类别上均属于缓冲区溢出，是其他两项工作支持检测的漏洞类型）。实验所用计算机的处理器为 Intel Core i7-4790 3.60GHz，内存 32GB，硬盘 1TB，GPU Geforece GTX 1070 Ti。

4.1 数据集

数据集包含了 CWE-120（Buffer Copy without Checking Size of Input），CWE-121（Stack-based Buffer Overflow）和 CWE-122（Heap-based Buffer Overflow）3 类漏洞，相关方法数统计如表 1 所列。

表 1 方法统计

Table 1 Functions statistics

类别	数量
方法总数	35374
含漏洞方法数	13875
不含漏洞方法数	20627
筛除方法数（过长过短）	872

训练集和测试集的划分采用十折交叉验证。

4.2 评估指标

实验以精确率、召回率和 F1 值 3 个衡量指标来评估结果的优劣。精确率，又名查准率，表示正确预测为含有漏洞的方法数占全部预测为含有漏洞的方法数的比例。召回率，又名查全率，表示正确预测为含有漏洞的方法数占全部实际含有漏洞的方法数的比例。F1 值是精确率和召回率的调和平均数。相关计算公式如下：

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

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

$$F1\text{-}measure=2\times\frac{precision\times recall}{precision+recall}$$

其中，TP 表示预测为存在漏洞且漏洞存在的方法数量，FP 表示预测为存在漏洞但漏洞不存在的方法数量，TN 表示预测为不存在漏洞且漏洞不存在的方法数量，FN 表示预测为不存在漏洞但漏洞存在的方法数量。

4.3 参数设置

模型的训练在 GPU 上进行，参数更新采用随机梯度下降法（Stochastic Gradient Descent，SGD），具体参数配置如表 2 所列。

表 2 参数配置

Table 2 Parameter configuration

参数	数值
样本数	34502
学习速率	0.001
批处理大小	128
词汇表大小	4332
嵌入向量维度	200
隐含层节点数	128
神经网络层数	2
最大梯度	5
Dropout	0.8

4.4 实验结果

为了进一步验证本文方法的有效性，本文基于上述数据集，运用十折交叉验证将其与现有的两项工作 VulDeePecker[21] 和 AE-KNN[26] 进行对比实验。实验结果如表 3 所列。

表 3 对比结果

Table 3 Comparative results

（单位：%）

算法	精确率	召回率	F1
VulDeePecker	91.7	82.2	86.7
AE-KNN	93.1	87.8	90.4
本文方法	95.3	93.5	94.4

实验结果显示，本文方法的精确率和召回率最高达到了 95.3% 和 93.5%，召回率与 VulDeePecker 相比更是高出了 11 个百分点。VulDeePecker 在数据预处理时，主要提取源代码中的 API 和库方法调用，生成 code gadgets。然而，该过程只针对 API 和库方法调用做提取，丢失了一些代码信息，导致其在召回率上的表现稍显逊色。

与 AE-KNN 相比，本文方法在准确率和召回率上分别高出了约 2 和 6 个百分点。AE-KNN 提取方法的 API 序列并量化为特征向量，然后对特征向量进行聚类，提取每一类中异常值排序高的样本函数以匹配漏洞库。因此，该方法的表现很大程度上依赖于其制定的漏洞库，并没有彻底解放人力。相比之下，本文方法不需要领域专家的参与，能够自动化地从漏洞样本中提取一些隐含的漏洞模式，省时省力，且具备良好的可扩展性。

5 有效性威胁

本文通过程序静态分析技术提取源代码的特征，并在此基础上使用 BiLSTM 神经网络学习漏洞模式。该方法以数据为驱动，因此对于不同的漏洞类型，能否找到与之匹配的、准确可靠的大型标记数据集是影响本文方法有效性的一个因素。

此外，不同类型的漏洞模式不同，学习该模式所依赖的程序特征也有所差别。本文在特征提取阶段尽可能地融合多种程序特征，以期许该方法在检测其他类型的漏洞时能够延续良好的表现。

结束语 为检测软件中隐含的大量漏洞，本文将代码的语句以及语句间的数据流和控制流依赖抽象为一种特征表示，并采用基于 Attention 的 BiLSTM 神经网络学习且训练出相应的漏洞分类模型。实验在 CWE120，CWE121 和 CWE122 这 3 种漏洞的小规模数据集上取得了 95.3% 的精确率和 93.5% 的召回率，优于现有的基于机器学习的漏洞检测方法。

基于该实验的后续工作主要分为两个方面：1)继续扩展数据集以包含更多的漏洞种类，验证该方法对其他漏洞类型的适应性；2)尝试在现有工作方法的基础上引入程序切片，将样本的粒度从方法级别缩小至程序片段，从而能够实现漏洞定位。

参 考 文 献

[1] GHAFFARIAN S M, SHAHRIARI H R. Software vulnerability analysis and discovery using machine-learning and data-mining techniques[J]. ACM Computing Surveys, 2017, 50(4): 1-36.

[2] US-CERT[OL]. http://us-cert.gov.

[3] 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. Paris, France: IEEE, 2010.

[4] WOO M, CHA S K, GOTTLIEB S, et al. Scheduling black-box mutational fuzzing[C]// Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security-CCS '13. New York: ACM Press, 2013.

[5] American fuzzy lop[OL]. http://lcamtuf.coredump.cx/a?/.

[6] WANG T L, WEI T, GU G F, et al. TaintScope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection[C]// 2010 IEEE Symposium on Security and Privacy. Oakland: IEEE, 2010.

[7] BÖHME M, PHAM V T, ROYCHOUDHURY A. Coverage-based greybox fuzzing as Markov chain[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16). Vienna, Austria. New York: ACM Press, 2016.

[8] RAWAT S, JAIN V, KUMAR A, et al. VUzzer: Application-aware Evolutionary Fuzzing[C]// NDSS. 2017.

[9] MOLNAR D A. Automated Whitebox Fuzz Testing[C]// Network & Distributed System Security Symposium. DBLP, 2011.

[10] BABIĆ D, MARTIGNONI L, MCCAMANT S, et al. Statically-directed dynamic automated test generation[C]// Proceedings of the 2011 International Symposium on Software Testing and Analysis-ISSTA'11. New York: ACM Press, 2011.

[11] NEUHAUS S, ZIMMERMANN T, HOLLER C, et al. Predicting vulnerable software components[C]// Proceedings of the 14th ACM conference on Computer and communications security-CCS'07. New York: ACM Press, 2007.

[12] YAMAGUCHI F, GOLDE N, ARP D, et al. Modeling and discovering vulnerabilities with code property graphs[C]// 2014 IEEE Symposium on Security and Privacy. San Jose, CA: IEEE, 2014.

[13] CHANDRAMOHAN M, XUE Y X, XU Z Z, et al. BinGo: cross-architecture cross-OS binary search[C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016). New York: ACM Press, 2016: 678-689.

[14] XU Z Z, CHEN B H, CHANDRAMOHAN M, et al. SPAIN: security patch analysis for binaries towards understanding the pain and pills[C]// 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). Buenos Aires: IEEE, 2017.

[15] LI Z, ZOU D Q, XU S H, et al. VulPecker: an automated vulnerability detection system based on code similarity analysis[C]// Proceedings of the 32nd Annual Conference on Computer Security Applications. 2016: 201-213.

[16] KIM S, WOO S, LEE H, et al. VUDDY: a scalable approach for vulnerable code clone discovery[C]// 2017 IEEE Symposium on Security and Privacy (SP). San Jose: IEEE, 2017.

[17] 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.

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

[19] RUSSELL R, KIM L, HAMILTON L, et al. Automated vulnerability detection in source code using deep representation learning[C]// 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). Orlando: IEEE, 2018.

[20] HARER J A, KIM L Y, RUSSELL R L, et al. Automated software vulnerability detection with machine learning[J]. arXiv: 1803.04497, 2018.

[21] LI Z, ZOU D Q, XU S H, et al. VulDeePecker: a deep learning-based system for vulnerability detection[C]// Proceedings 2018 Network and Distributed System Security Symposium. Reston, VA: Internet Society, 2018.

[22] LI Z, ZOU D, XU S, et al. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities[J]. arXiv: 1807.06756, 2018.

[23] ANTLR4[OL]. https://github.com/antlr/antlr4.

[24] Common Weakness Enumeration[OL]. https://cwe.mitre.org.

[25] Software Assurance Reference Dataset of National Institute of Standards and Technology[OL]. https://samate.nist.gov/SARD.

[26] LI Y C, HUANG R, LAI F G, et al. Open source software vulnerability detection method based on deep clustering[J]. Application Research of Computers, 2020, 37(4): 1107-1110, 1114.

GONG Kou-lin, born in 1995, postgraduate, is a member of China Computer Federation. His main research interests include software evolution analysis and mining software repositories.

ZHOU Yu, born in 1981, Ph.D, professor, is a member of China Computer Federation. His main research interests include software evolution analysis, mining software repositories, software architecture, and reliability analysis.