基于图网络的 Java 反序列化漏洞检测方法

胡　飞[1],陈　昊[2],王　媛[1],弋　雯[1],胡　颖[3],刘宝英[1]
(1. 西北大学 信息科学与技术学院,陕西 西安 710100;
2. 中国劳动关系学院,北京 100048;
3. 新华社技术通信局,北京 100803)

摘　要:Java 反序列化漏洞由于其很容易被非法利用,已经成为目前最具威胁的软件漏洞之一。在开发过程中,事先对软件所使用的第三方公共组件库进行检测,提前发现并防御潜在的反序列化漏洞尤为重要。目前已有的反序列化漏洞检测,主要有基于规则匹配和基于污点分析两种检测方法,前者采用白名单或者黑名单的方法无法发现未知的反序列化漏洞,而后者因其对漏洞调用链检测能力有限,故漏报和误报率高。为了弥补已有方法的缺陷,提出了一种基于图网络的 Java 反序列化漏洞调用链检测方法 SerialFinder,该方法利用图结构充分表达反序列化漏洞调用链的语义信息,训练图同构网络模型,进而可以检测潜在的反序列化漏洞调用链。SerialFinder 在多个第三方组件库进行验证,与业界最先进的 Java 反序列化漏洞调用链检测方法 Gadget Inspector 进行对比,结果表明,SerialFinder 在三个公共组件库上的平均命中率为 64%,比 Gadget Inspector 高 31%。

关键词:漏洞检测;图数据库;Java 反序列化;图神经网络;调用链
中图分类号:TP391　　**文献标识码**:A　　**文章编号**:1673-629X(2023)05-0122-08
doi:10.3969/j.issn.1673-629X.2023.05.019

Call Chain Detection Method for Java Deserialization Vulnerability Based on Graph Network

HU Fei[1], CHEN Hao[2], WANG Yuan[1], YI Wen[1], HU Ying[3], LIU Bao-ying[1]
(1. School of Information Science & Technology, Northwest University, Xi'an 710100, China;
2. China University of Labor Relations, Beijing 100048, China;
3. Technical Bureau of Xinhua News Agency, Beijing 100803, China)

Abstract: The Java deserialization vulnerability has become one of the most threatening software vulnerabilities due to its easy exploitation. During the development process, it is particularly important to detect the third-party public component library used by the software in advance, and to detect and defend against potential deserialization vulnerabilities in advance. At present, the existing deserialization vulnerability detection mainly includes two detection methods based on rule matching and based on taint analysis. The former cannot find unknown deserialization vulnerabilities by using whitelist or blacklist methods, while the latter has a high rate of false negatives and false positives due to its limited ability to detect vulnerability call chains. In order to make up for the shortcomings of existing methods, we propose a call chain detection method SerialFinder based on graph network for Java deserialization vulnerabilities. The method uses the graph structure to fully express the semantic information of the deserialization vulnerability call chain, trains the graph isomorphic network model, and then can detect the potential deserialization vulnerability call chain. SerialFinder is verified in multiple third-party component libraries and compared with Gadget Inspector, the industry's most advanced Java deserialization vulnerability call chain detection method. The results show that SerialFinder has an average hit rate of 64% on the three public component libraries, which is 31% higher than Gadget Inspector.

Key words: vulnerability detection; graph database; Java deserialization; graph neural network; call chain

收稿日期:2022-05-04　　**修回日期**:2022-09-07
基金项目:陕西省国际合作计划重点项目(2020KWZ-013);中国劳动关系学院一般项目(20XYJS007)
作者简介:胡　飞(1996-),男,硕士研究生,研究方向为深度学习和漏洞检测;通信作者:陈　昊(1976-),女,博士,副教授,研究方向为无线传感网和网络信息安全。

0　引　言

以 5G 通信、物联网、大数据[1]为代表的先进技术已经融入人们的生活当中,计算机软件在人们的生活中随处可见,软件已经成为经济、军事、文化和社会的重要引擎。近年来网络攻击[2]、软件崩溃[3]、信息泄露[4]等事件屡见不鲜,给国家和社会造成了巨大的经济损失,因此软件安全问题不可忽视,软件漏洞的检测也迫在眉睫[5]。

最近几年来美国国家漏洞数据库[6](National Vulnerability Database,NVD)的软件漏洞数量持续增加,据 NVD 报告,2019 年统计的漏洞数量达到了 140 000 条,这一数量相比 1999 年增长了将近 10 倍。软件的快速发展和扩大同样也引起了更多的软件漏洞产生。由于诸多原因,在软件开发过程中,无论是软件的复杂度还是开发者协作,又或者是程序员在编程时缺乏安全的编码意识,都会引发软件安全问题,因此软件漏洞的产生无可避免。

序列化是软件开发过程中一种用于捕获程序状态的机制,用于持久化运行时状态或跨进程边界的过程调用。它涉及到将内部运行时表示转换为二进制或字符数据流并返回[7]。简单来说,序列化就是将对象的状态信息持久化存储,比如保存到文件或者数据库中用于持久化,反之,反序列化就是将序列化的数据反向重新构建对象,恢复对象的状态信息。早在 2006 年反序列化漏洞就已经出现,但由于造成的影响较小,因此未引起大多数安全研究人员的注意[8]。2015 年 Java 反序列化漏洞迎来了关注,Apache Commons Collections 库中报告发现一个漏洞,包含该漏洞的开源组件 Commons Collections 库被各种主流的 Web 应用所使用,影响范围十分广泛,从此之后,有关反序列化漏洞的研究开始流行起来,该漏洞使得许多企业系统可被远程代码执行利用。2016 年,反序列化漏洞攀升至 OWASP[9](Open Web Application Security Project)前十的第八位。从图 1 可以看到,自 2015 年,每年报告在通用漏洞披露[10](Common Vulnerabilities and Exposures,CVE)中的 Java 反序列化漏洞数量一直不低于 30 条,在 2016 年达到了最高,高达 65 条。

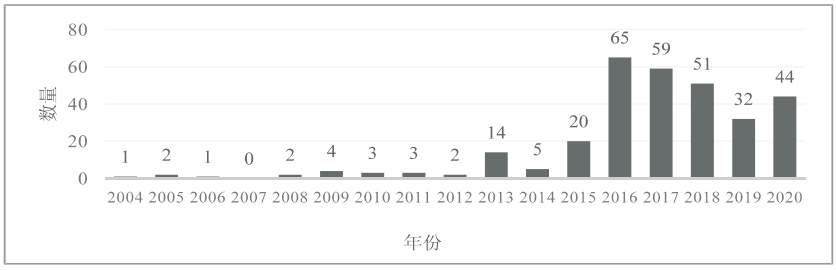

图 1　Java 反序列化漏洞 CVE 数量统计图

许多面向对象编程的语言都支持序列化,大多数这些语言的标准库中都包含了对序列化的支持。在 Java 语言中,本地序列化对象是用非语言构造的,例如绕过构造函数,对象状态是由反序列化的数据恢复的。反序列化漏洞除了在 Java 中存在,在脚本语言 PHP、Python 中同样也存在。反序列化漏洞相较于其他漏洞,如 SQL 注入[11](SQL Injection)、拒绝服务[12](Denial Of Service,DOS)、缓冲区溢出[13](Buffer Overflow)等,它通常所需要的权限较低,但是却能获得较高的管理权限,因而造成的风险危害大,具有很高的威胁性。

目前,人工智能领域的快速发展,机器学习和深度学习技术的不断深入,为软件安全漏洞领域提供了新的检测思路,研究人员应用这些技术和方法取得了较好的效果。针对 Java 反序列化漏洞[14],该文旨在运用图神经网络检测 Java 反序列化漏洞调用链。图神经网络是基于图结构知识训练神经网络模型,能够捕获漏洞更深层的语义和语法信息,为漏洞检测研究领域提供了方向。通过分析 Java 反序列化调用链的模式和行为,使用图神经网络深度学习模型来识别反序列化漏洞,从而检测出 Java 反序列化漏洞调用链。

1　相关工作

针对 Web 漏洞,许多学者依据上述漏洞检测方法进行了研究。Jovanovic[15]通过静态源代码分析,来解决 Web 应用程序易受攻击的问题,使用上下文敏感的数据流分析来检测程序中的漏洞。他提出静态分析[16]工具 Pixy,用于检测 PHP 脚本程序中的跨站脚本攻击(Cross Site Script Attack,XSS)和 SQL 注入漏洞[11]。Livshits[17]提出对象污点传播的方法,从源头(Source)到汇聚点(Sink)进行分析来检测漏洞。Viktoria 等人[18]使用动态分析的方法观察 Web 应用程序的运行过程定制正确的行为规范,借助符号执行[19]的方法识别出违反正确规范的程序路径,从而检测应用程序的逻辑错误。Li 等人[20]提出了检测 C 语言程序漏洞的检测系统 VulDeePecker,通过提取程序源码的函数块进行数据流分析,对函数体切片,使用自然语言处理领域的向量嵌入技术 Word2vec[21]对代码

切片编码,将编码的向量输入到双向长短记忆神经网络(Bidirectional Long Short Term Memory Networks, BiLSTM),选择二分类监督学习来检测漏洞。然而,上述的这些漏洞检测方法在应用到 Java 反序列化漏洞检测时效果不佳。

Apache Commons Collections 中的反序列化漏洞在 2015 年被公布,研究人员描述了 Java 反序列化漏洞调用链的执行过程[22]。为了更好地挖掘反序列化漏洞,Federico Dotta 在 2016 年开发出 Java Deserialization Scanner[23]开源插件,该插件可以用于 Burp Suite 上检测和利用反序列化漏洞。Carettoni 则提出了针对 Java 反序列化漏洞的防御方案 Serial Killer,即采用白名单或者黑名单的方式,手动控制添加禁止危险的工具类[24]。还有一些类似 Java Deserialization Scanner 的一系列小插件 Freedy[25]、SuperSerial[26]等等,它们依托 NCC 组织开发的 Burp 插件,作为 Burp 的扩展来利用反序列化漏洞。这些反序列化漏洞检测工具可以加深漏洞安全研究人员对反序列化漏洞调用链执行过程的理解,给反序列化漏洞调用链的检测提供了参考,但缺乏自动化检测的能力,并且需要人工专家进行审计,因而存在较大的局限性。

2018 年,Netflix 高级安全软件工程师 Ian Haken 提出了一种基于 Java 字节码分析的反序列化漏洞调用链检测方法 Gadget Inspector[27]。该工具基于静态分析的方法首先将 Java 开发工具包(Java Development Kit,JDK)中所使用到的 jar 包以及第三方公共组件库解析成字节码,确立调用链的 Source 和 Sink,应用污点分析的技术判断入口点到汇聚点的连通路径是否可达,从而检测可以利用的反序列化调用链。该方法是目前使用广泛的反序列化漏洞调用链检测方法,但是由于采用基于静态污点分析的技术,Gadget Inspector 存在一定的缺陷,由于搜索路径没有覆盖到完整的子类和接口实现类,存在较高的误报率和漏报率。

2 基于图网络的反序列化漏洞调用链检测方法

本节将介绍基于图网络的反序列化漏洞调用链检测方法 SerialFinder。文章首先对反序列化漏洞调用链的检测流程进行概述,然后介绍多关系边图神经网络模型学习,包括多边关系图构建、学习节点和边的向量化表示、基于图多边调用关系的聚合,最后介绍图神经网络模型的训练和检测。

2.1 反序列化漏洞调用链检测框架

该文利用图神经网络构建反序列化漏洞检测模型。该方法的主要思想是将已知的包含反序列化漏洞调用链的公共组件库作为输入,然后进行数据收集工作。对于数据收集过程,首先会将公共组件库解析保存至图数据库,然后基于多关系边提取调用链子图并对数据标注,完成训练数据的收集工作。对收集到的调用链子图进行特征提取和多关系边调用图构建,使用工具对节点和边进行向量初始化。将调用链子图处理后的向量信息输入到 GIN 进行训练,保存验证集上效果最好的模型。最后,使用训练好的图网络模型对未知的公共组件库进行检测,检测未知公共组件库中的反序列化漏洞调用链。

图 2 详细描述了 SerialFinder 检测框架。

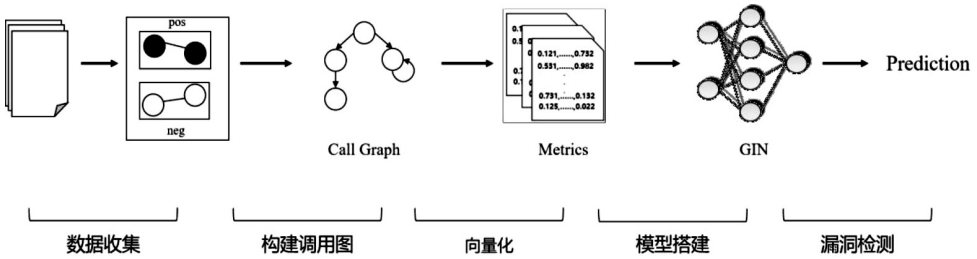

图 2 SerialFinder 检测框架

2.2 多边关系图构建

依据调用链节点的执行顺序构建调用图,构建完好的多关系边调用图如图 3。其中每一个调用链子图对应一个多关系边调用图。调用图为有向图,图包含方法节点和邻接关系边。函数调用图可用于描述方法之间的一般调用,多关系边调用图在函数调用图的基础上进行了改进,扩展为由 6 种关系边组成的调用图。为了学习到每个方法节点的向量表示,在图网络模型学习阶段,在每个节点上增添 Self-Loop 边,使边的数量加倍,这些 Self-Loop 边有利于不同调用图之间的信息传播。下面分别对 6 种关系依赖边进行介绍。

(1)ALIAS 边:保存 Java 多态机制中的动态链接关系。

(2)InvokeStatic 边:静态方法调用指令,保存方法节点对静态方法的调用关系。

(3)InvokeSpecial 边:保存类中私有方法节点、构造方法节点的方法调用关系。

(4)InvokeVirtual 边:动态分配调用指令,保存方

法节点对实例方法的调用关系。

(5) InvokeInterface 边:接口调用指令,保存接口引用对实例方法的调用关系。

(6) InvokeDynamic 边:方法句柄调用指令,保存方法句柄发起的调用关系。

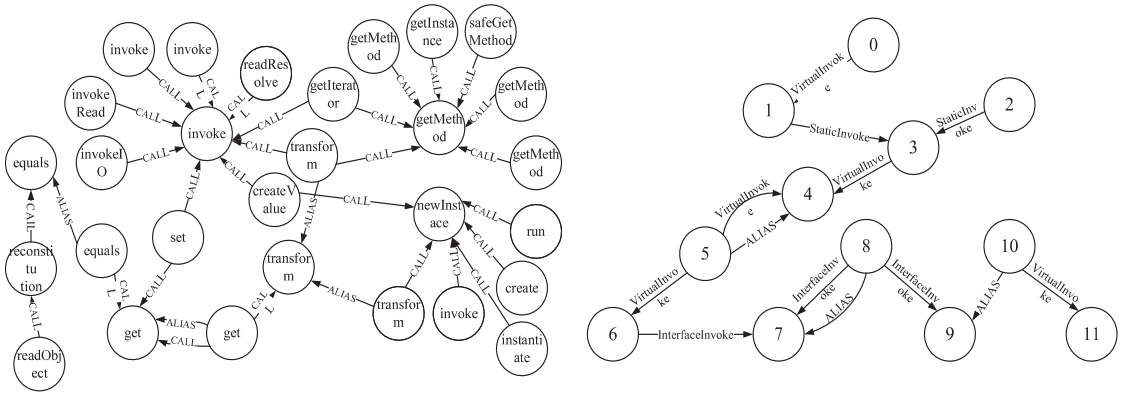

图 3　调用链属性图和多关系边调用图示例

2.3　学习节点和边的向量化表示

为了将调用链子图转化为图神经网络模型可以输入的上下文表示向量,首先将提取出的方法节点信息和边的关系信息进行向量化。具体操作步骤如下:

(1)构建方法节点的语料库。将方法节点的方法签名进行预处理,分离出方法签名的信息,包括方法所属的类、方法的返回类型、方法名、方法的参数列表。基于所有存储在图数据库中的方法节点,收集了数十万方法节点的方法签名用作构建方法节点的语料库。

(2)构建边关系的语料库:方法节点和边关系为调用链子图的不同信息,因而对于边关系,单独将边的关系提取出来构建语料库。其中包含 ALIAS、InvokeStatic、InvokeSpecial、InvokeInterface、Invoke - Virtual、InvokeDynamic 这 6 种调用边关系。

(3)训练 word2vec 模型:选择了一种较为流行的向量化工具—word2vec,它是语言模型的一种,能从大量的文本语料中以无监督学习的方式来学习知识模型,广泛应用于自然语言处理领域。该文选择 CBOW 模型用来学习方法节点和边关系在词典中的向量表示。CBOW 的处理方式是在已知词 w_t 的上下文,例如上下文为 w_{t-3}、w_{t-2}、w_{t-1}、w_{t+1}、w_{t+2}、w_{t+3} 的情况下预测当前词 w_t 的概率。CBOW 模型与神经网络十分类似,主要包含三层:首先是输入层,读取上下文的词向量;然后是隐藏层,对输入层的词向量进行累积求和;最后是输出层,原理上是一个哈夫曼树,基于 Softmax 函数来输出预测的概率。将收集到的方法节点的语料库输入到 CBOW 模型,模型对语料库中的词的向量进行随机初始化,然后基于随机梯度下降(Stochastic Gradient Descent,SGD)的学习方式更新模型的参数和词的向量,学习完成后可以得到一个词典,词典是一个词和词向量对应的映射表。通过学习得到的 CBOW 模型,可以将方法节点和边关系进行向量化。

(4)为了捕获调用链子图的完整信息,将调用链子图的方法节点和边关系经过 CBOW 模型嵌入向量后按照调用顺序进行拼接,其中每一个调用链子图对应一个 $N \times 100$ 维的向量。

2.4　基于图多边调用关系的聚合

GAT、GCN 等流行的图神经网络大多采用邻居节点聚合的方式,该文所使用的图同构网络也采用了这种设计,但在此基础上进行了改进。图 4 展示了图同构网络的相邻节点聚合方式。所有的节点均有自己的特征向量,图中展示了 3 种类型的节点,分别是 1 号节点、2 号节点、3 号节点。其中 1 号节点位于最外层,1 号节点与 2 号节点相连,它们将自己的特征向量以一种特定规则聚合计算合并到 2 号节点,同样 2 号节点聚合特征向量到 3 号节点。每个节点由其邻居节点的状态和信息所得到。每一次迭代都会进行若干次聚合,在重复了固定次数的迭代之后,计算聚合了相邻节点信息的节点的特征向量。

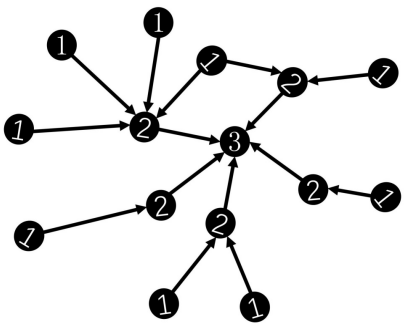

图 4　多关系相邻节点聚合图

该文对多关系边调用图的聚合进行改进,在计算相邻节点聚合时,采用 MLP 机制。图 5 为多关系边调用图的聚合方式,其中在第一层不采用 MLP,因为节点的初始特征向量不用于 MLP 的计算。使用多层感

知机(MLPs)对函数建模,符合节点的单射性计算。

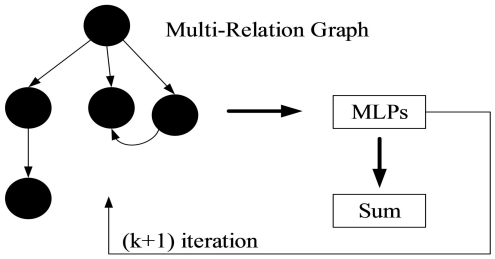

图 5 多关系边调用图聚合图

2.5 模型训练和检测

相比于传统机器学习需要对特征进行人工定义,该文采用的图同构网络对嵌入的向量进行自动化学习。图同构网络在流行的图神经网络上进一步革新,保证图结构的单射性,提升了图神经网络检测的准确率。

模型的参数对于模型效果有很大影响,为了提升模型的准确率,该文选择一个基准数据用于模型的参数调优。由于 GPU 资源条件限制,使用多个批次进行网络的训练。对于深度学习网络,模型需要依靠损失函数计算损失值,从而对模型的参数进行更新。该文使用交叉熵损失函数,因为它十分适合图的分类任务。神经网络的神经元需要通过激活函数激活,因而激活函数的选择也十分重要。在图网络的初始层选择 Tanh 激活函数,在隐藏层采用 Tanh 激活函数,在消息传递层选择 Relu 激活函数,使用不同的激活函数能提升模型的学习能力。对于图神经网络,考虑到相邻节点的聚合方式,选择 Sum 的聚合函数,考虑相邻所有节点的特征向量。对于图的二分类问题,模型的最后一层需要将高维向量映射到[0,1]空间,Softmax 函数提供了这种能力。选择 Softmax 作为图同构网络的最后一层,经过 Softmax 层,可以得到图网络模型对图的一个分类结果,即 0 或 1,其中 0 代表图是不包含漏洞的,1 代表图包含漏洞,即存在反序列化漏洞。

在模型检测阶段,首先对公共组件库生成对应的图数据库进行保存,然后依据图搜索算法提取并筛选由 Source 到 Sink 的调用链子图。对调用链子图提取静态特征和多关系边调用图,经向量化嵌入后输入到训练好的图网络模型进行预测,从而检测出反序列化漏洞调用链。

将调用链子图作为图神经网络的输入,当模型给出的预测结果为 1 时,则表明调用链子图存在反序列化漏洞调用链。该检测方案可以定位到调用链子图,实现反序列化漏洞的细粒度定位。

3 实验与分析

3.1 实验数据分布

为了训练一个有效的深度学习模型来实现对 Java 反序列化漏洞调用链的检测,文中的训练数据来自于现实世界真实存在的反序列化漏洞。在实验中,从多个受影响的公共组件库中进行数据搜集工作,涉及到的公共组件库有 Commons-Collections-3.1、Commons-Collections4-4.0、FileUpload1.3.1、XStream1.4.15、Hibernate5.0.7.Final 等。由于实际存在漏洞的调用链极其缺乏,设计了基于多关系边提取调用链子图的方法,对包含漏洞调用链的子图进行提取扩充,因为调用链是已经被证实存在反序列化漏洞的,因而包含漏洞调用链的子图即为存在漏洞的数据。该文按照反序列化漏洞调用链所属的组件不同进行分类,表 1 给出了具体的数据信息。其中 Commons-Collections、JDK、Hibernate 都是基于 JDK 的反序列化 API 引发漏洞,可以归结为 JDK API 类,而 XStream 反序列化依靠 fromXML( ) 函数进行,因而将 XStream 单独归为一类。调用链主要来自于 4 个公共组件库,从原始调用链数目来看导致反序列化漏洞的调用链实际上非常少,这对于深度学习模型的训练来说是十分欠缺的。基于文中的调用链子图提取方法,提取出的调用链子图的总数量为 113 670,其中正负样本集比例为 1∶1,漏洞调用链子图和非漏洞调用链子图数目均为 56 835。收集到的调用链子图数据,为图神经网络模型提供了充足的数据基础。

表 1 调用链子图数据分布

公共组件库	原始调用链	调用链子图	危险调用链子图
Commons-Collections	7	48 600	24 300
JDK	3	17 730	8 865
Hibernate	2	14 260	7 130
XStream	5	33 080	16 540

3.2 实验环境配置

文中图同构网络模型训练的硬件配置为:Ubutun18.04 64 位、NVIDIA GeForce RTX 2080 GPU、Intel Core i9-9940 CPU。基于深度学习平台 TensorFlow 来搭建 GIN 漏洞检测模型,在模型训练前,将收集到的训练数据按照 8∶1∶1 的比例分离,训练集占 80%,验证集和测试集各占 10%。在训练过程中,保存在验证集上评估效果最好的模型用于与其他

工具进行比较。为了对公共组件库进行字节码层面的解析,借助了 SOOT 这个工具来进行解析,同时使用 Neo4j 图数据库来保存类和方法对应的节点以及边的关系。在向量化调用链子图的节点时,使用 word2vec 这个自然语言模型来处理,同时将每个节点的向量设置为 100 维向量,对图中的节点数量没有进行限制。在得到最终的模型之前,选择样本集来对模型的超参数进行自动化调优,其中每个 batch 的最大节点数范围为(32、64、128),隐藏层的层数范围为(64、128、256),epoch 设置为 100,并且在 25 轮训练中没有得到更好效果模型的时候进行自动停止。

3.3　对比工作及评价指标

将 SerialFinder 与最近的反序列化漏洞调用链检测方法 Gadget Inspector 进行比较。该工具是基于静态分析的方法,它首先将 Java 环境 JDK 中所使用到的 jar 包以及第三方工具类解析成字节码,确立调用链的入口点(Source)和汇聚点(Sink),应用污点分析的技术判断入口点到汇聚点的连通路径是否可达,从而检测出可以利用的反序列化漏洞调用链。选择 3 个已知的存在反序列化漏洞的公共组件库 Commons-Collections-3.1、Hibernate5.0.7.Final、Commons-Collections4-4.0,因为这些组件库被反序列化漏洞研究人员广泛研究和利用。基于这三个公共组件库对比两种方法的漏洞检测能力,最后分析检测出的调用链是否为漏洞调用链。

该文选择十折交叉验证的方法来评估模型的效果,即将数据分为相同数量的 10 份,每次训练选择其中的 9 份用于模型训练,另外 1 份用于模型的测试阶段。使用该标准方法对于模型预测的评估有很强的泛化能力。在进行实验结果分析中,选择了广泛使用的度量指标,分别有:准确率(Accuracy)、查准度(Precision)、召回率(Recall)、F1-score、误报率(False Positive Rate,FPR)。同时,在交叉验证中对上述的度量指标计算各自的几何平均值,几何平均值与算术平均值相比更加可靠。

3.4　实验结果与分析

(1)不同边关系选择对比。

为了评估基于不同关系边构图对 SerialFinder 检测能力的影响,将 SerialFinder 与基于两种关系边构图的方法进行对比。其中基于两种关系边构图的方法,只包含 ALIAS 边和 Call 边,Call 边为方法节点间的通用调用关系。选取的方法包含 6 种关系边 ALIAS、InvokeStatic、InvokeSpecial、InvokeInterface、InvokeVirtual、InvokeDynamic。实验在收集到的三种类型的数据上进行对比。为了方便实验描述,将基于两种关系边构图的方法称作 Two-relation。表 2 为两种处理方法的 Accuracy、Precision、Recall、F1-score 在三种类型数据上的平均值对比结果。

表 2　SerialFinder 与 Two-Relation 在 5 个指标上的平均值　%

检测方法	Accuracy	Precision	Recall	F1-score	FPR
Two-Relation	84.6	80.0	92.3	85.7	23.1
SerialFinder	97.0	94.0	100.0	96.9	5.9
比较结果	+12.4	+14.0	+7.7	+11.2	-17.2

从表 2 可以得出,SerialFinder 在三种类型数据上的平均 Accuracy、Precision、Recall、F1-score 分别达到 97.0%、94.0%、100.0%、96.9%;通过比较发现,在前四个指标上,SerialFinder 分别比 Two-Relation 高出了 12.4 百分点、14.0 百分点、7.7 百分点以及 11.2 百分点,并且在 FPR 指标上 SerialFinder 比 Two-Relation 低了 17.2 百分点。因此,可以得知依赖 6 种关系边构图的方法能提升 SerialFinder 的检测性能。

(2)不同图神经网络对比。

为了评估图同构网络 GIN 对 SerialFinder 检测能力的影响,选择图神经网络研究领域中使用比较多的图神经网络模型进行对比,对比的模型包括门控图神经网络 GGNN、图卷积神经网络 GCN、图注意力网络 GAT,实验数据选择该文收集到的三种类型数据。在使用 GCN、GAT、GGNN 进行网络训练之前,首先会提取调用链子图并使用 word2vec 对图中的节点和边进行向量化。在三种类型的数据上进行实验,结果如表 3~表 5 所示。

表 3　不同图神经网络在 JDK API 数据集上的 Accuracy、Precision、Recall、F1-score、FPR　%

图神经网络	Accuracy	Precision	Recall	F1-score	FPR
GGNN	81.8	89.7	83.9	86.7	23.1
GCN	71.9	71.9	84.0	83.6	16.8
GAT	90.9	89.5	94.4	91.9	13.3
GIN	96.9	94.4	100.0	97.1	6.7

表 4 不同图神经网络在 XStream 数据集上的 Accuracy、Precision、Recall、F1-score、FPR %

图神经网络	Accuracy	Precision	Recall	F1-score	FPR
GGNN	88.9	90.9	90.9	90.9	14.3
GCN	75.8	84.6	64.7	73.3	12.5
GAT	91.5	84.6	100.0	91.7	16.0
GIN	97.2	94.7	100.0	97.3	5.6

表 5 不同图神经网络在 JDK 和 XStream 混合数据集的 Accuracy、Precision、Recall、F1-score、FPR %

图神经网络	Accuracy	Precision	Recall	F1-score	FPR
GGNN	90.6	92.9	80.0	88.9	13.0
GCN	78.1	85.7	50.0	63.2	5.6
GAT	90.6	84.2	100.0	91.4	18.7
GIN	96.9	92.9	100.0	96.3	5.3

从表中的数据可以计算得到,GIN 在三种类型数据上的 F1-score 平均值为 96.9%,分别较 GGNN(88.8%)、GCN(73.4%)、GAT(91.7%)高出了 8.1 百分点、23.5 百分点、5.2 百分点,意味着 GIN 在保证检测的准确率高的同时,可以检测出更多的漏洞样本。在 FPR 指标上,GIN 在三种类型数据上的平均 FPR 值为 5.9%,相较于 GGNN(16.8%)、GCN(11.6%)、GAT(16%)分别低了 10.9 百分点、5.7 百分点、10.1 百分点,这意味着 GIN 网络检测出漏洞样本的误报率很低。SerialFinder 使用的 GIN 网络在三种类型数据上均表现出最好的性能。

(3)不同 Java 反序列化漏洞检测方法对比。

为了评估 SerialFinder 对 Java 反序列化漏洞调用链的检测能力,实验选择 Java 语言常见的三个公共组件库:Commons-Collections-3.1、Commons-Collections4-4.0、Hibernate5.0.7.Final。在这三个公共组件库上展开调用链的检测工作,与 Gadget Inspector 进行对比。对识别为漏洞的调用链,还需要对其进行人工分析,才能确定是否为恶意的漏洞调用链。本节采用命中率来比较两种方法检测反序列化漏洞的能力,命中率即漏洞调用链占全部调用链的比值,见如下公式。

$$\text{Shoot} = \frac{\text{Vulnerable Gadget Chain}}{\text{ALL Gadget Chain}}$$

SerialFinder 和 Gadget Inspector 在三个公共组件库上的命中率如表 6,在 Commons-Collections-3.1、Commons-Collections4-4.0、Hibernate5.0.7.Final 上,SerialFinder 的命中率分别为 67.0%、75.0%、50.0%,相比 Gadget Inspector 分别高出了 17.0 百分点、25.0 百分点以及 50.0 百分点,平均高出了近 31 百分点。可以得知,SerialFinder 表现出更好的反序列化漏洞调用链检测能力。

表 6 SerialFinder 与 Gadget Inspector 在 3 个公共组件库上的命中率 %

检测方法(平均命中)	CommonCollections3.1	CommonCollections4.0	Hibernate5.0.7
Gadget Inspector(33%)	50.0	50.0	0
SerialFinder(64%)	67.0	75.0	50.0
比较结果	+17.0	+25.0	+50.0

(4)检测性能对比。

为了评估 SerialFinder 和 Gadget Inspector 的检测性能,选择 Commons-Collections-3.1、Commons-Collections4-4.0、Hibernate5.0.7.Final 三个公共库进行检测,记录多次测试的平均检测时间。

SerialFinder 的图网络模型训练需要若干个小时,但是训练开销是一次的,不需要重新训练和部署,因而可以忽略不计。表 7 显示了 SerialFinder 和 Gadget Inspector 在三个公共组件库上的检测时间,可以计算得到 SerialFinder 对三个公共库的平均检测时间为 21 s,Gadget Inspector 对三个公共库的平均检测时间为 43 s,SerialFinder 相比 Gadget Inspector 的检测时间低了 22 s。因此可以得知文中方法有着更高的检测效率。

表 7 SerialFinder 和 Gadget Inspector 在三个公共库上的运行时间 s

检测方法	Commons-Collection-3.1	Commons-Collections4-4.0	Hibernate5.0.7.Final
Gadget Inspector	42	41	47
SerialFinder	17	20	25

4　结束语

在目前的漏洞检测技术中,静态分析的方法需要人工专家来制定规则,不但要耗费大量的人力,而且在很大程度上依赖于定义规则的准确性。动态分析的方法需要程序完好的运行,具备程序运行的可行性条件,对于不同的软件需要准备不同的运行环境,有较高的时间和运行成本。深度学习的方法,可以避免人工专家定义特征和规则,能有效缩小代码审计的范围,实现漏洞检测的自动化。该文提出基于图神经网络检测反序列化漏洞的方法 SerialFinder,实现了一种自动化的 Java 反序列化漏洞调用链检测原型系统。为了评估 SerialFinder 的检测能力,选择 3 个已知的公共组件库与 Gadget Inspector 进行对比,实验结果表明,SerialFinder 检测出了更多的反序列化漏洞调用链,并且有更高的命中率;在时间性能的比较中,SerialFinder 的运行时间更短,有更高的检测效率。

参考文献:

[1] OUSSOUS A,BENJELLOUN F Z,LAHCEN A A,et al. Big data technologies:a survey[J]. Journal of King Saud University-Computer and Information Sciences,2018,30(4):431-448.

[2] HANSMAN S,HUNT R. A taxonomy of network and computer attacks[J]. Computers & Security,2005,24(1):31-43.

[3] HUANG Shih-Kun, HUANG Min-Hsiang, HUANG Po-Yen,et al. Software crash analysis for automatic exploit generation on binary programs[J]. IEEE Transactions on Reliability,2014,63(1):270-289.

[4] ISSA I, WAGNER A B, KAMATH S. An operational approach to information leakage[J]. IEEE Transactions on Information Theory,2019,66(3):1625-1657.

[5] MCGRAW G. Software security[J]. IEEE Security & Privacy,2004,2(2):80-83.

[6] US-CERT Security Operations Center. Nationalvulnerability database[EB/OL]. 2022. https://nvd. nist. gov.

[7] HERLIHY M P,LISKOV B. A value transmission method for abstract data types[J]. ACM Transactions on Programming Languages and Systems,1982,4(4):527-551.

[8] SCHNEIDER C. Java deserialization security FAQ[EB/OL]. 2020. https://christian - schneider. net/JavaDeserialization-SecurityFAQ. html.

[9] WICHERS D, WILLIAMS J. Owasp top-10 2017[J]. OWASPFoundation,2017,3:4-5.

[10] The MITRE Corporation. Common vulnerabilities and exposures[EB/OL]. 2020. http://cve. mitre. org.

[11] 陈小兵,张汉煜,骆力明,等. SQL 注入攻击及其防范检测技术研究[J]. 计算机工程与应用,2007,43(11):150-152.

[12] 林　栋. 拒绝服务攻击(DoS)的攻与防[J]. 广东通信技术,2003,23(4):26-28.

[13] 蒋卫华,李伟华,杜　君. 缓冲区溢出攻击:原理,防御及检测[J]. 计算机工程,2003,29(10):5-7.

[14] 杜笑宇,叶　何,文伟平. 基于字节码搜索的 Java 反序列化漏洞调用链挖掘方法[J]. 信息网络安全,2020,20(7):19-29.

[15] JOVANOVIC N,KRUEGEL C,KIRDA E. Static analysis for detecting taint-style vulnerabilities in web applications[J]. Journal of Computer Security,2010,18(5):861-907.

[16] 夏一民,罗　军,张民选. 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.

[17] LIVSHITS V B,LAM M S. Finding security vulnerabilities in java applications with static analysis[C]//USENIX security symposium. Baltimore:USENIX,2005:18.

[18] FELMETSGER V,CAVEDON L,KRUEGEL C,et al. Toward automated detection of logic vulnerabilities in web applications[C]//19th USENIX security symposium (USENIX Security 10). Washington:USENIX,2010:10.

[19] 叶志斌,严　波. 符号执行研究综述[J]. 计算机科学,2018,45(6A):28-35.

[20] 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 (NDSS). San Diego:Internet Society,2018.

[21] CHURCH K W. Word2Vec[J]. Natural Language Engineering,2017,23(1):155-162.

[22] GABRIEL L,CHRIS F. AppSecCali2015:marshalling pickles [EB/OL]. 2020. https://frohoff. github. io/appseccali - marshalling-pickles.

[23] FEDERICODOTTA, GOLDSTEIN J, VERES - SZENTKIRÁLYI A. Java-deserialization-scanner[EB/OL]. 2021. https://github. com/federicodotta/Java-Deserialization-Scanner.

[24] MUÑOZ A, SCHNEIDER C. Serial killer: silently pwning your Java endpoints[C]//RSA conference. San Francisco: Rohit Ghai,2016.

[25] VAN DER BAAN S,DALILI S,JOHNSTON P. freddy[EB/OL]. 2021. https://github. com/nccgroup/freddy.

[26] DirectDefense Inc. SuperSerial[EB/OL]. 2020. https://github. com/DirectDefense/SuperSerial.

[27] HAKEN I. Automated discovery of deserialization gadget chains[R]. The United States: Black Hat Briefings and Trainings,2018.