doi :10.3969/j.issn.1671-1122.2022.09.005

基于数据流追溯的空指针引用挖掘系统

文伟平，刘成杰，时林
（北京大学软件与微电子学院，北京 100080）

摘　要：空指针异常引用是系统运行过程中的一种常见问题，该问题会引起程序崩溃或者异常退出，同时攻击者也可以利用空指针解引用来完成任意读写操作，导致信息泄露。Java 作为一种广泛使用的语言，也存在空指针引用问题，主要原因是对引用变量的指向检查不足。文章提出一种基于数据流追溯的空指针引用检测系统，并设计了静态分析工具 jvd。该工具通过特化追踪空指针在容器中的传播，使得空指针变量不会在容器中传播丢失，在中间语言 Jimple 层面上完成检测并覆盖多种空指针容器传播场景，有效降低复杂场景下的漏报率。在 Juliet Test Suite 的 CWE476 号测试集上，将文章所设计的 jvd 与 SpotBugs、Infer 等工具进行对比实验。实验结果表明，jvd 能够在多种空指针传播场景下使用，在高精度场景下能够取得比其他工具更好的效果。

关键词：数据流分析；空指针解引用；Jimple；容器传播

中图分类号：TP309　**文献标志码：**A　**文章编号：**1671-1122（2022）09-0040-06

中文引用格式：文伟平，刘成杰，时林 . 基于数据流追溯的空指针引用挖掘系统 [J]. 信息网络安全，2022，22（9）：40-45.

英文引用格式：WEN Weiping, LIU Chengjie, SHI Lin. A Null Pointer Reference Mining System Based on Data Flow Tracing[J]. Netinfo Security, 2022, 22(9): 40-45.

A Null Pointer Reference Mining System Based on Data Flow Tracing

WEN Weiping, LIU Chengjie, SHI Lin
（*School of Software and Microelectronics*, *Peking University*, *Beijing* 100080, *China*）

Abstract: Null pointer dereference is a common defect in programming, which often causes the program crash or abnormal exit. At the same time, attackers can also use null pointer dereference to complete arbitrary read and write operations, leading to information disclosure. Java is a widely used language, and also suffers from null pointer dereference due to insufficient checks on dereference. In order to avoid the potential risk, this paper proposed a null pointer dereference detection system based on data flow analysis and designed a static analysis tool jvd. This tool implemented analysis on Jimple and covered multiple container propagation cases, especially in containers by special treatment, which effectively reduced the false negative rate in complex scenarios. This paper completed the experiment and compared jvd with several popular tools like SpotBugs and Infer on CWE476 test dataset in Juliet Test Suite, which shows that jvd could be used in multiple null pointer transmission and achieved excellent performance in high accuracy situation.

Key words: data flow analysis; null pointer dereference; Jimple; container propagation

收稿日期：2022-05-31
基金项目：国家自然科学基金 [61872011]
作者简介：文伟平（1976—），男，湖南，教授，博士，主要研究方向为系统与网络安全、大数据与云安全、智能计算安全；刘成杰（1998—），男，湖南，硕士研究生，主要研究方向为软件安全、漏洞挖掘、入侵检测；时林（1998—），男，山东，硕士研究生，主要研究方向为漏洞挖掘、软件安全防护。
通信作者：文伟平 weipingwen@pku.edu.cn

0 引言

空指针解引用是程序运行过程中的一种常见问题[1]，其根本原因是运行时发生引用的变量或者指针未被分配有效的内存地址空间，产生空指针解引用异常，导致程序提前终止，甚至系统产生雪崩式崩溃。同时，空指针引用缺陷可能被恶意攻击者利用，诱发信息泄露、DDoS攻击等诸多不良后果。相比于语法类型问题，空指针引用缺陷更难以被常规的审计工具、漏洞挖掘工具发现，因为其仅在程序运行状态下被触发，并且空指针的产生与引用之间可能存在复杂的代码逻辑关系，所以对此类缺陷的排查很困难。

Java是目前很受欢迎的一种编程语言，在程序框架、移动应用软件、互联网中间件等领域得到广泛应用。在使用Java的过程中，由于数据范围检查和代码测试的不完备性，Java程序中也存在空指针引用，并且可能在程序线上运行时触发空指针异常，导致服务进入不可用状态。

为了避免发生空指针引用，当前业界采用程序分析的方式提前检测潜在的缺陷，程序分析通常使用静态分析、动态分析或者两者结合的方式挖掘空指针引用缺陷。静态分析[2]主要在程序运行之前进行，通过对程序的语法结构、代码规范、数据范围等进行分析来发现程序中的潜在缺陷或者漏洞，但是准确率相对较低。动态分析在程序运行时进行，通过观察针对特定输入时程序响应的行为来判断程序中是否存在逻辑，以此确定程序的安全性和鲁棒性。例如，将fuzzing技术应用于漏洞挖掘[3]，但是代码覆盖率偏低并且非常依赖于测试用例。本文针对静态分析的不足，基于数据流分析方法设计实现了一个空指针异常引用检测系统。该系统在Jimple代码[4]层面完成检测，并且覆盖私有方法调用、容器传播和系统调用等多种场景。

1 研究现状

当前，系统主要通过对程序中的变量进行数据流分析来完成程序静态分析。王锐强[5]提出一种利用判断逻辑信息进行空指针的检测，但是在解引用点和判空点之间缺少数据流分析，因此存在一定的精度误差。NANDA[6]等人提出一种上下文敏感的空指针引用检测方式，首先在取得程序控制流图的前提下，收集发生指针解引用的语句；然后进行反向数据传播，并且在数据流分析过程中进行一定程度上的路径探索；最后在约束条件到达定值或者分析到达程序入口时，给出指针引用的安全性结果。但该方法在反射调用、并发性运行方面未做相应处理。马森[7]等人采用守卫标注的方式通过值依赖关系完成空指针引用检测，针对每个程序，首先计算该程序中的各个数据依赖关系；然后生成各个指针的依赖关系图，在图的节点之间建立“守卫”，即节点之间的到达约束条件；最后依靠程序值依赖分析图结合到达条件来检测空指针，在大型程序中的适用性更好。白杨[8]等人通过传统静态分析和符号执行相结合的方式实现一种支持多敏感类型的空指针引用检测，一方面通过符号执行对不可达路径进行剪枝，避免无效引用分析；另一方面在函数分析退出时完成分支合并，从而降低时间开销，进一步提高检测效率，但准确率不高。DUAN[9]等人使用动静结合的方式动态生成异常调用栈信息，能够反向分析程序，并定位空指针产生的语句，但无法确定其方法的通用性。JIN[10]等人针对二进制程序进行空指针检测，该方法改进了指针别名分析和恶意源识别但是检测工具存在过污染的情况，分析效果极度依赖于BAP（Binary Analysis Platform）二进制分析平台[11]，并且会对不可达路径仍然执行操作，最终造成误报。

综上所述，针对Java的空指针引用检测研究不能解决检测过程中面临的各种情况，当前业界提出的方法具有一定局限性。本文在研究现有工具的不足后，对一些已有的方法进行改进，本文所提方法可以覆盖更广泛的空指针传递情况，通过设计jvd有效完成Java空指针引用的检测。

2 系统整体架构

空指针检测系统架构如图1所示，本文提出的空指针检测系统主要包括字节码转换、前置分析、数据

流分析和缺陷报告4个模块。字节码转换模块主要将相应的Jar包文件的字节码转换成Jimple语言表示，通过该过程保留必要的原始信息，去除不必要的信息，为后续检测提供中间代码。前置分析模块主要抽取待分析程序中的控制流图、调用流图，同时完成已有指针的指向分析和约束求解。数据流分析模块首先针对程序中存在的每个指针语句，利用前置模块提供的上下文信息完成反向传播；然后针对每条被执行的语句进行数据流转换，并生成对应的约束条件；最后对约束条件进行逻辑表达式约简。缺陷报告模块负责统计引用的定位信息，包括指针引用所属类、所属函数名、发生引用的行号、指针引用类型、危险性等，最后生成空指针检测报告。

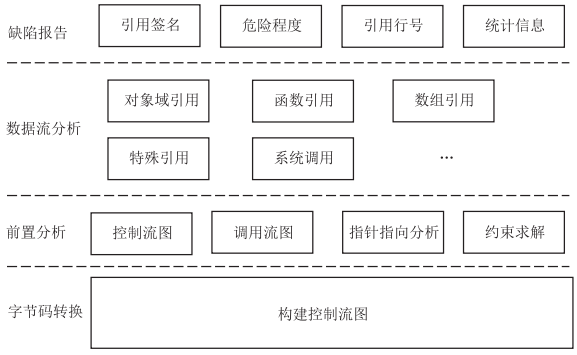

图 1 空指针检测系统架构

3 系统功能

空指针检测系统整体分析流程如图2所示，Java程序被字节码转换模块接收，经过分析后对缺陷进行报告并输出结果。本文的工作主要体现在前置分析模块和数据流分析模块，其中前置分析模块为后续分析提供指针引用、控制流图、调用流图等信息，数据流分析模块对多种引用以及系统调用进行数据流分析。

3.1 字节码转换模块

系统接收的输入文件是一个Java可执行程序，字节码转换模块负责解析程序、读取主类等元信息，并使用Soot库[12]完成字节码的转换。在转换过程中，选择应用程序模式，通过规则将.class文件中的字节码表示转换为Jimple语言表示，并把可执行程序元信息

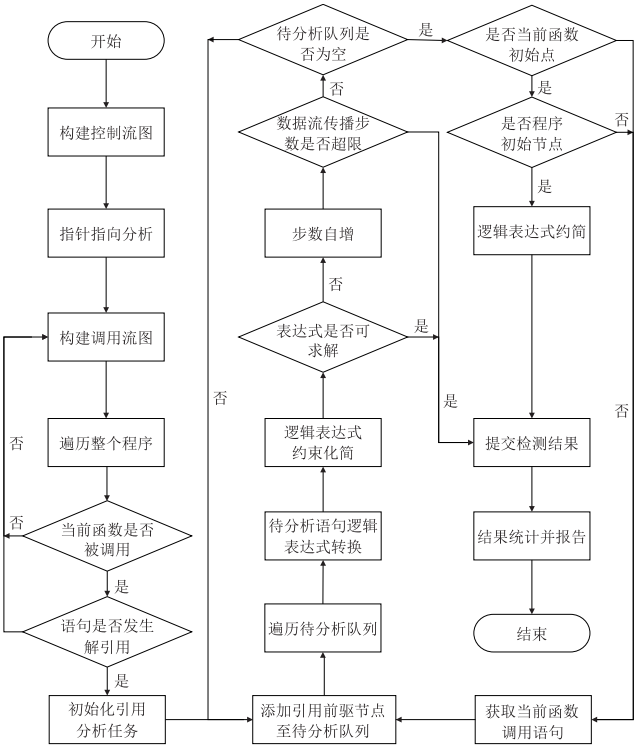

图 2 空指针检测系统整体分析流程

存储在数据结构中。字节码翻译前后对比如表1所示，本文在Soot源码的基础上进行修改，更改字节码转换逻辑，保留发生过程中空指针引用时Jimple代码层面的原始语义，提升了检测准确率。

表 1 字节码翻译前后对比

语句类型	语句内容
Java 语句	String s = null
	s.length()
原始 Jimple 语句	r1 = new java.lang.NullPointerException
	specialinvoke r1.<java.lang.NullPointerException: void <init>(java.lang.String)>(“This statement would have triggered an Exception: $stack2 = virtualinvoke s.<java.lang.String: int length()>()")
	throw r1
修改后 Jimple 语句	r1 = null
	virtualinvoke r1.<java.lang.String: int length()>()

3.2 前置分析模块

前置分析模块负责分析Jimple代码中的元信息，从Jimple代码中构建调用流图、控制流图。同时，该模块负责遍历整个程序和添加所有指针解引用语句，并排除黑名单内容。如果某个函数不在调用流图中存在入度，表明该函数不被实际调用，因此一定不存在

空指针引用异常，即不对该函数内部引用展开后续分析，从而得到待分析指针列表。

3.3 数据流分析模块

数据流分析模块是空指针引用检测算法的核心，针对单个引用完成数据流分析，并给出待分析引用的安全性判断及元信息。针对前置分析模块得到的待分析指针，进行如下分析。

1）初始化约束条件，例如，“object.call();”表达式建立“null == object”初始约束条件。

2）获取当前语句的所在函数并加入调用栈，判断是否存在循环调用的过程，若存在循环调用，则中断分析过程。

3）获取当前函数的控制流图，选择当前语句的前一节点，并将该节点添加至待分析队列。遍历待分析队列，不断对当前语句完成数据流转换，将当前语句的前置节点添加至队列中，直至队列为空。

4）数据流转换本质上是数据流传播的过程，需要针对不同Jimple语句类型分别完成处理。假设约束条件为“null == obj1”，当前执行语句为“obj1=obj2”，当前语句为赋值类型，将约束条件转换为“null==obj2”。具体转换过程如表2所示，其中初始约束条件为null==a。

表 2 约束条件转换过程

语句	转换方式	转换后	语句解释
a = b	替换	null==b	赋值
a = null	替换	null==null	空赋值
a = @parameter	替换	null==@parameter	参数传递
a = new A()	替换	null==new Expr	构造对象
a = x.y	替换	null==x.y	域引用
if(c==d)	添加	null==a&&c==d	条件语句
goto:	保留	null==a	跳转语句
a = b+c	删除	\	运算语句
a = x[index]	替换	null==x:index	数组运算
return a	保留	null==a	返回值
a = call()	递归处理	视情况而定	函数调用
a = v.call()	替换	null==v:tag	序列容器
a = m.call()	替换	null==m:key	关联性容器
throw r1	保留	null==a	抛出异常
a = (cast_expr)b	替换	null==b	类型转换

5）待分析队列为空后，此时当前语句到达函数入口点，对约束条件完成逻辑约简。如果逻辑约简结果为“是”，表明存在空指针引用，将缺陷元信息添加至列表中。如果结果为“否”，表明在传递过程中存在内存分配，不存在空指针引用。如果结果为“不确定”，则需要继续执行下一步工作。

6）在数据流分析到达函数入口后，寻找该函数的调用语句，用实参替换@parameter变量，并将现有的上下文信息复制到调用语句处，从该语句开始继续通过控制流图寻找前驱节点，实现上下文敏感的空指针检测。

7）根据终止条件决定是否终止传播过程。如果数据流分析到达程序初始点后，仍然得不到确切的结果，则认为在静态分析当前代价的前提下，无法得到确切的分析结果。如果单次数据流分析的步数或者耗时超过阈值，则终止传播过程。如果程序中包含循环函数调用，则认为程序中存在调用问题，也会终止传播过程。如果以上传播过程在非确切逻辑值的前提下终止，那么系统会返回不确定性分析结果。

在数据流转换过程中，本文对容器类型的转换做了一些工作。当空指针在容器中传播时，可能该空指针变量仅占据容器的某一个位置，空指针传入和流出都只与该内存块有关。如果不对容器中的污染位置加以区分，则会出现欠污染或者过污染情况。本文依赖于容器实例、容器标签、数据下标等信息对空指针进行唯一定位，防止在容器传播过程中出现空指针扩散和丢失情况。空指针传播支持的容器类型如表3所示。

表 3 空指针传播支持的容器类型

容器类型	容器操作
Array	Array[index] = val
	val = Array[index]
Vector	void add(int index, E element)
	boolean add(E element)
	boolean remove(E element)
	E remove(int index)
LinkedList	void add(int index, E element)
	boolean add(E element)
	E remove(int index)
	E remove()
	boolean offer(E element)
HashMap	V put(K Key, V Value)
	V get(K Key)
Stack	E push(E)
	E pop()
	E peek()

结合表2和表3可以看出，通过对容器及其存取方法的区别处理，利用为对象加标签的方式，将空指针

在容器中的传播过程表示出来，避免空指针丢失或者扩散。

3.4 缺陷报告模块

当待分析指针列表分析完成后，缺陷报告模块收集引用的分析结果、引用签名、引用行号等元信息，对检测结果进行分类并生成空指针检测报告。

4 实验及分析

为了验证本文系统在空指针引用检测方面的有效性，选取 Juliet 1.3 Test Suite[13] 数据集为实验对象，该数据集由美国国家安全局安全软件中心研发，是一个包含上千种测试程序的系统性测试集，涵盖 C/C++、Java 等多种语言，包含 100 多种类型的程序错误。本文选取针对 Java 的编号为 CWE476_NULL_Pointer_Dereference 的缺陷数据集进行实验，该数据集包括约 280 个测试类、180 个空指针引用样例等。本文对测试样例按照传播方式进行划分，分为过程内和过程间两个类别，在每个类别中又对数据类型、触发方式等做了详细区分，并分别进行测试。

本文分别选择 SpotBugs[14-15] 和 Infer[16] 两种空指针检测工具完成对比实验。两者都是常用的空指针引用缺陷检测工具，其中 SpotBugs 的前身是 FindBugs[17-18]，SpotBugs 从缺陷模式的角度出发，对代码编写特征进行匹配进而检测是否存在缺陷。Infer 是 Facebook 公司开源的静态程序分析工具，用于在发布前对代码进行分析，从而找出潜在的问题。Infer 在静态分析过程中加入数据流追踪功能，进一步增强对缺陷的检测能力。过程内检测结果如表 4 所示，过程间检测结果如表 5 所示。

从表 4 和表 5 可以看出，在过程内的样例检测中，jvd 和两个对比工具的漏报率很低，几乎都为 0，检测效果比较接近。这是因为过程内的空指针传播相对简单，可变情况较少。在过程间的样例检测中，SpotBugs 的漏报率约为 60%~70%，即使加入数据流分析，Infer 平均漏报率也高于 40%，而 jvd 的漏报率为 5% 左右。这是因为 jvd 在实现过程中，完成精细化的过程间空指针传播，尤其细化了空指针在序列式容器和关联式容器中的传播，使得在回溯分析指针引用时，空指针变量不会丢失，从而大幅度降低误报率。实验结果表明，jvd 在高精度空指针解引用挖掘场景下的表现明显好于其他工具。

表 4 过程内检测结果

工具	空指针类型	真实报告数 / 个	缺陷总数 / 个	漏报率	用时 /s
SpotBugs	String	17	18	5.5%	5
	StringBuilder	17	18	5.5%	6
	Integer	17	18	5.5%	5
	array	17	18	5.5%	5
	check	17	17	0	6
	binary	17	17	0	6
Infer	String	17	18	5.5%	32
	StringBuilder	17	18	5.5%	32
	Integer	17	18	5.5%	32
	array	17	18	5.5%	32
	check	17	17	0	32
	binary	17	17	0	32
jvd	String	18	18	0	368
	StringBuilder	18	18	0	327
	Integer	18	18	0	393
	array	18	18	0	275
	check	17	17	0	10
	binary	17	17	0	13

表 5 过程间检测结果

工具	空指针类型	真实报告数 / 个	缺陷总数 / 个	漏报率	用时 /s
SpotBugs	Integer	6	19	68.4%	5
	StringBuilder	6	19	68.4%	7
	Array	7	18	61.1%	5
	String	6	19	68.4%	5
Infer	Integer	11	19	42.1%	12
	StringBuilder	11	19	42.1%	12
	Array	11	18	38.9%	12
	String	9	19	52.6%	12
jvd	Integer	18	19	5.3%	219
	StringBuilder	18	19	5.3%	228
	Array	17	18	5.6%	186
	String	18	19	5.3%	214

5 结束语

本文针对传统空指针静态挖掘方法的不足，设计并优化了基于数据流分析的空指针挖掘方法。通过追踪空指针在容器中的传播，使得空指针变量不会在容器中传播丢失，从而覆盖更多程序传播路径，在复杂场景下有效降低漏报率。通过在 Juliet Test Suite 数据集上的实验以及与其他工具对比表明，jvd 的检出效果优

于其他工具。同时本文还存在需要改进的方面，主要是针对文件流等变量传播不敏感，下一步将进行改进。

参考文献：

[1] MITRE. 2022 CWE Top 25 Most Dangerous Software Weaknesses[EB/OL]. (2022-05-03)[2022-05-12]. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.

[2] ZHIOUA Z, SHORT S, ROUDIER Y. Static Code Analysis for Software Security Verification: Problems and Approaches[C]//IEEE. 2014 IEEE 38th International Computer Software and Applications Conference Workshops. New York: IEEE, 2014: 102-109.

[3] SHAO Lin, ZHANG Xiaosong, SU Enbiao. New Method of Software Vulnerability Detection Based on Fuzzing[J]. Application Research of Computers, 2009, 26(3): 1086-1088.

邵林，张小松，苏恩标 . 一种基于 fuzzing 技术的漏洞发掘新思路 [J]. 计算机应用研究，2009，26（3）：1086-1088.

[4] VALLEE-RAI R, HENDREN L J. Jimple: Simplifying Java Bytecode for Analyses and Transformations[EB/OL]. (2004-01-01)[2022-03-25]. https://www.researchgate.net/publication/243776080_Jimple_Simplifying_Java_Bytecode_for_Analyses_and_Transformations.

[5] WANG Ruiqiang. Null Pointer Reference Pattern Detection Based on Judgment Logic[D]. Beijing: Beijing University of Posts and Telecommunications, 2015.

王锐强 . 基于判断逻辑的空指针引用模式检测 [D]. 北京：北京邮电大学，2015.

[6] NANDA M G, SINHA S. Accurate Interprocedural Null-Dereference Analysis for Java[C]//IEEE. 2009 IEEE 31st International Conference on Software Engineering. New York: IEEE, 2009: 133-143.

[7] MA Sen, ZHAO Wen, XI Xiangyu, et al. Null Pointer Dereference Detection Based on Value Dependences Analysis[J]. Acta Electronica Sinica, 2015, 43(4): 647-651.

马森，赵文，习翔宇，等 . 基于值依赖分析的空指针解引用检测 [J]. 电子学报，2015，43（4）：647-651.

[8] BAI Yang, WANG Yuping. Multiple Sensitive Static Method of Detecting Null Pointer Reference Bug[J]. China Sciencepaper, 2014, 9(10): 1131-1136.

白杨，王瑀屏 . 一种多敏感空指针引用错误的静态检测方法 [J]. 中国科技论文，2014，9（10）：1131-1136.

[9] DUAN Jing, JIANG Shujuan, YU Qiao, et al. An Automatic Localization Tool for Null Pointer Exceptions[J]. IEEE Access, 2019(7): 153453-153465.

[10] JIN Wenhui, ULLAH S, YOO D, et al. NPDHunter: Efficient Null Pointer Dereference Vulnerability Detection in Binary[J]. IEEE Access, 2021(9): 90153-90169.

[11]BRUMLEY D, JAGER I, AVGERINOS T, et al. BAP: A Binary Analysis Platform[C]//Springer. International Conference on Computer Aided Verification. Heidelberg: Springer, 2011: 463-469.

[12] VALLÉE-RAI R, CO P, GAGNON E, et al. Soot: A Java Bytecode Optimization Framework[C]//ACM. CASCON First Decade High Impact Papers(CASCON'10). New York: ACM, 2010: 214-224.

[13] NIST. Juliet 1.3 Test Suite: Changes from 1.2[EB/OL]. (2018-06-14)[2022-05-21]. https://doi.org/10.6028/NIST.TN.1995.

[14] Spotbugs. Find Bugs in Java Programs[EB/OL]. (2022-03-22)[2022-05-08]. https://spotbugs.github.io/.

[15] TOMASSI D A. Bugs in the Wild: Examining the Effectiveness of Static Analyzers at Finding Real-World Bugs[C]//ACM. 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2018: 980-982.

[16] Facebook. A Tool to Detect Bugs in Java and C/C++/Objective-C Code Before it Ships[EB/OL]. (2021-11-23)[2022-01-12]. https://fbinfer.com/.

[17] University of Maryland. FindBugs-Find Bugs in Java Programs[EB/OL]. (2021-05-11)[2022-01-12]. http://findbugs.sourceforge.net/.

[18] AL-AMEEN M N, HASAN M M, HAMID A. Making Findbugs More Powerful[C]//IEEE. 2011 IEEE 2nd International Conference on Software Engineering and Service Science. New York: IEEE, 2011: 705-708.