第 34 卷 第 21 期 计 算 机 工 程 2008 年 11 月
Vol.34 No.21 Computer Engineering November 2008
·安全技术· 文章编号：1000—3428(2008)21—0124—02 文献标识码：A 中图分类号：TP311

基于数据流的脆弱性静态分析

匡春光，陈 华，张鲁峰
(北京系统工程研究所，北京 100101)

摘 要：为提高 Java 软件的安全性，针对 Java 程序的脆弱性分析问题，提出一种基于数据流的感染分析法，阐述了具体思路和实现步骤。依据该方法实现的分析系统能有效分析出 Java 字节码程序中存在的 XPath 注入、SQL 注入等脆弱性，结果证明了基于数据流的感染分析法的正确性和可行性。

关键词：脆弱性；控制流；数据流

Static Analysis of Vulnerability Based on Dataflow

KUANG Chun-guang, CHEN Hua, ZHANG Lu-feng
(Beijing Institute of System Engineering, Beijing 100101)

【Abstract】 In order to improve the security of Java software, this paper suggests a dataflow based taint analysis method to solve the problem of vulnerability analysis related to Java programs. The idea of the dataflow based taint analysis method is introduced and the process of taint analysis method is presented. Analysis system built according to the method can effectively find the vulnerabilities in Java ByteCode programs, such as XPath injection, SQL injection, etc. Results verify the correctness and validity of the method.

【Key words】 vulnerability; control flow; dataflow

1 概述

Java 语言由于其平台无关性而得到越来越广泛的应用。相应地，Java 程序的安全性也越来越受到重视，其脆弱性是引发 Java 程序安全问题的主要因素，使用脆弱性静态分析技术尽早发现 Java 程序中存在的脆弱性可以避免许多不必要的损失。

本文介绍一种针对Java程序的基于数据流的感染分析法。Java程序中一些特定的脆弱性是由于程序接收了用户输入信息后，又利用这些信息访问敏感系统资源(如XML文件中的数据[1-2]、数据库中的数据、特权代码等)而产生的，因为在这种情况下，用户可以通过输入恶意信息，达到访问不该访问的系统资源的目的。此方法的正确性和有效性已通过编程实现得到验证。

2 基于数据流的感染分析法的思路

目前已有的脆弱性静态分析技术主要有语法模式匹配法、利用类型限定符的感染分析法、模型检测法[3]和自动定理证明法。语法模式匹配法的缺点是会产生大量虚报错误。利用类型限定符的感染分析法的最大缺陷是需要人工在源代码中增加与分析有关的注释，实用性不是很强。模型检测法存在状态爆炸问题，不太适用于大型程序。自动定理证明法的缺点是分析过程需要人工干预，自动化程度不高。本文提出的基于数据流的感染分析法能实现全自动化，且能达到较高的准确率。

为了分析程序是否接收用户的输入信息，并把这些信息传播到访问敏感系统资源的代码处，应解决以下 3 个问题：

(1)必须确定被分析的程序是否接收用户信息。Java 程序接收用户信息是由一些特定的方法实现的，这些方法包括 getParameter, getParameterMap 和 getQueryString 等。通过查找程序中是否使用这些方法可判断程序是否接收用户信息。

(2)必须确定被分析的程序是否访问敏感系统资源。与接收用户信息类似，Java 程序访问敏感系统资源也由一些特定的方法实现，这些方法包括 executeQuery, evaluate 和 selectNodes 等。通过查找程序中是否使用这些方法可以判断程序是否访问敏感系统资源。

(3)必须分析用户的输入信息是否会传播到访问敏感系统资源的代码处。即使程序接收了用户信息，也访问了敏感系统资源，但如果它们之间没有联系，就不能说明程序中存在特定的脆弱性。只有它们之间存在联系，才能说明程序中可能存在特定的脆弱性。这是最关键的问题，也是最难解决的问题。

最简单的情况是，程序接收用户信息后，把用户信息保存在一个变量中，访问敏感系统资源时直接使用这个变量。但这种情况很少发生。通常的情况是，程序接收用户信息后，把用户信息保存在一个变量(设为 a)中，然后利用这个变量构造其他变量(设为 b)，这种构造可能会发生多次，生成变量 $c, d, e\cdots$，访问敏感系统资源时使用最后一次构造生成的变量(设为 e)，用户输入的信息并未直接用来访问敏感系统资源，但这些信息通过传播，仍然会影响敏感系统资源的访问。

3 感染分析方法

本文提出一种逆向捕捉的方法来捕捉这种传播过程，即先分析程序访问敏感系统资源时使用了哪些变量，然后分析这些被使用的变量在哪儿赋值，对这些被使用的变量赋值时又使用了哪些变量，依此类推，一步步往前找，如果其中有一个变量是由接收用户信息的方法赋值的，则说明程序中可能存在特定的脆弱性。如果变量的使用和赋值之间存在一一

作者简介：匡春光(1971－)，女，副研究员，主研方向：计算机安全；陈 华，研究员；张鲁峰，副研究员

收稿日期：2007-12-24 **E-mail：**flintflint@sina.com

对应的关系，上文的方法就可以较准确地确定程序中是否存在特定的脆弱性。但当程序对同一个变量进行多次赋值时，该方法的准确性就会大大降低，因为在没有更多信息的情况下，只能假设变量的每一次赋值都会影响该处变量的使用，但实际上有且只有一次会影响该处变量的使用。

为了提高准确性，有必要对程序进行变换，使其具有变量的一次使用只有唯一的一次赋值和其对应的特性，即单赋值特性，因为本文提出的方法是一种静态分析法，所以只要具有静态单赋值特性即可，把程序转换为静态单赋值表示就可以达到此目的。把程序转换为静态单赋值表示需要对变量重命名，使变量的每次使用和对应的赋值具有同一名称，不对应的赋值具有不同名称。

确定变量的使用和程序中哪一次赋值具有对应关系需要对程序进行控制流分析，构建程序的控制流图 CFG，即把程序分割成多个基本块，基本块在 CFG 中也叫节点，一个节点由一组顺序执行的语句组成，节点和节点之间则存在分支转移关系，一个节点可能有多个后继节点，也可能有多个前驱节点。对一个节点中一个变量的一次使用，如果在同一节点中，在使用该变量前对该变量赋值，该变量此处的使用-赋值关系是很明显的。如果在同一节点中，在使用该变量前未对该变量赋值，该变量此处的使用-赋值关系就复杂了，该节点的所有前驱节点中对该变量的赋值都可能影响该变量此处的使用。

为了达到变量的一次使用只有唯一的一次赋值和其对应的目的，可以在该节点的第一个语句前对该变量增加一次赋值，赋值时使用 n 个参数，每一个参数代表该变量从该节点的前驱节点引进的一次赋值，此类赋值可能存在于前驱节点中，也可能存在于前驱的前驱中，甚至更多级的前驱中，显然，这些前驱可能重合，这就意味着赋值时使用的 n 个参数中可能有 2 个甚至更多个源于同一个前驱，如果 n 个参数都源于同一个前驱，则该节点中就不需要增加对该变量的赋值了。此外，增加的赋值也不一定要在该节点中，应放置在会聚节点中，会聚节点是指有多个前驱节点的节点。

对一个变量，应在给它赋值的所有节点的支配前沿中增加赋值。支配前沿的确定需要利用 CFG 中各个节点的支配节点集和最邻近支配节点等信息。

4 感染分析实现步骤

针对 Java 程序的基于数据流的感染分析法的实现主要包括以下几部分：

(1)构建程序的 CFG

构建程序的CFG时，先把程序分成若干基本块，每一个基本块是CFG中的一个节点，如果控制可以从一个节点A流向另一个节点B，则在CFG中增加一条从A到B的边，A成为B的前驱节点，B成为A的后继节点[4-5]。把程序分成若干基本块时，先确定哪些指令是 1 个基本块的第 1 条指令(top)，程序的第 1 条指令是，分支指令的潜在目标指令也是。从一个top到下一个top的前一条指令(bottom)构成一个基 本块。

(2)分析 CFG 中各个节点的支配节点集

CFG 中节点 n 的支配节点是指从程序的入口到 n 的所有路径都经过的节点，所有支配节点的集合就构成支配节点集。显然，n 是它自身的支配节点，n 的所有前驱节点的支配节点集的交集中的节点也是 n 的支配节点。为了提高算法的效率，在计算 CFG 中各个节点的支配节点集之前，先计算 CFG 中节点的深度优先前序访问序列，计算各个节点的支配节点集时按此序列的先后顺序进行。

(3)分析 CFG 中各个节点的最邻近支配节点

CFG 中节点 n 的最邻近支配节点 id 是指满足如下条件的节点：1) id 是 n 的支配节点；2)$id \neq n$；3)不存在任一节点 m, $m \neq id$, $m \neq n$, id 支配 m, m 支配 n。直观地说，id 就是所有支配节点中级别最低的节点。分析 id 的算法可以借助冒泡排序算法的思想，先设支配节点集中的第一个节点为临时的最邻近支配节点 t，将 t 与支配节点集中的其他节点 r 比较，如果 t 不是 r 的支配节点，则继续往后比较，如果 t 是 r 的支配节点，则使 $t=r$，继续往后比较，最后得到的 t 就是 n 的最邻近支配节点。

(4)分析 CFG 中各个节点的支配前沿

节点 n 的支配前沿是一个节点集，节点集中的每一个节点 f 至少有一个前驱 p 被 n 支配，但 f 本身不被 n 支配，或者 $f=n$。实际上，n 的支配前沿可以由 2 部分组成：1)n 的部分后继节点，这些节点的最邻近支配节点不是 n；2)n 的最邻近被支配节点的支配前沿中的一部分节点，这些节点的最邻近支配节点不是 n。为提高算法的效率，在计算 CFG 中各个节点的支配前沿之前，先计算 CFG 中节点的深度优先后序访问序列，计算各个节点的支配前沿时按此序列的先后顺序进行。

(5)在程序中插入必要的变量赋值

为了达到变量的一次使用只有唯一的一次赋值和其对应的目的，对一个变量，应该在给它赋值的所有节点的支配前沿中增加赋值。因为增加赋值后，变量的赋值点可能又增加了，所以这一过程可能要反复多次。使用工作链的方法可以很大地提高算法的效率。工作链的方法如下：

1)把对该变量赋值的所有节点都存到工作链中。

2)如果工作链为空，算法结束，否则，从工作链中取出一个节点 n，如果某节点 f 是节点 n 的支配前沿中的节点，而且在节点 f 中还没有增加对该变量的赋值，则：在节点 f 中增加对该变量的赋值，赋值时使用的参数个数与节点 f 的前驱节点的个数相同，参数名暂时与该变量同名；标记在节点 f 中已增加对该变量的赋值了；如果在增加赋值前节点 f 中没有对该变量的赋值，则把节点 f 存入工作链中。

3)转到 2)。

(6)对变量重命名，生成程序的静态单赋值表示

在程序中插入必要的变量赋值后，变量的一次使用只有唯一的一次赋值与其对应，但变量的多次赋值使用的是同一变量名，这种使用—赋值关系不能直观地表示出来，为了直观地表示出这种使用—赋值关系，可以对变量重命名，对变量的每一次赋值都赋予一个新的名称，对应的使用点也赋予同样的名称。

(7)分析程序中是否存在特定的脆弱性

首先分析程序中是否有访问敏感系统资源的方法，如果有，则分析程序访问敏感系统资源时是否直接或间接使用了用户的输入信息，如果使用了，则说明程序中可能存在特定的脆弱性。

5 结束语

本文主要从基本思想和实现 2 个方面对基于数据流的感染分析法进行了阐述。基于数据流的感染分析法的主要原理

（下转第 128 页）