第34卷 第12期 计 算 机 工 程 2008年6月
Vol.34 No.12 Computer Engineering June 2008
·安全技术· 文章编号：1000—3428(2008)12—0157—03 文献标识码：A 中图分类号：TP309

软件安全漏洞的静态检测技术

张 林[1,2]，曾庆凯[1,2]
(1. 南京大学计算机软件新技术国家重点实验室，南京 210093；2. 南京大学计算机科学与技术系，南京 210093)

摘 要：软件安全漏洞问题日益严重，静态漏洞检测提供从软件结构和代码中寻找漏洞的方法。该文研究软件漏洞静态检测的两个主要方面：静态分析和程序验证，重点分析词法分析、规则检查、类型推导、模型检测、定理证明和符号执行等方法，将常用的静态检测工具按方法归类，讨论、总结静态检测技术的优势、适用性和发展趋势。
关键词：软件安全漏洞；静态分析；程序验证

Static Detecting Techniques of Software Security Flaws

ZHANG Lin[1,2], ZENG Qing-kai[1,2]
(1. State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093;
2. Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

【Abstract】 This paper summarizes two strategies of software security flaw detection, named static analysis and program verification. Several static detection methods such as lexical analysis, rule checking, type theory deduction, model checking, theorem proving, and symbol execution are also synthetically reviewed. It discusses the advantage, applicability and tendency of static detecting techniques.
【Key words】 software flaws security; static analysis; program verification

1 概述

漏洞指软件设计实现过程中被引入的、在数据访问或行为逻辑等方面的缺陷，它可能被攻击者利用从而使程序行为违背一定的安全策略[1-2]。

按照检测过程中是否需要执行程序的标准，软件安全漏洞检测技术分为：

(1)动态测试

在程序运行过程中注入测试数据，观察程序运行是否正常、输出是否符合程序意图，达到寻找程序漏洞的目的。动态测试仅关注程序运行的外部表现，因此，其定位不准确、漏报率高。

(2)静态检测

关注程序的代码，从程序代码的内部结构和特性上检测漏洞，适当地弥补了这一缺陷。

静态检测技术从早期的词法分析开始，出现了大量的方法。早期静态检测主要指静态分析[3]，随着形式化验证方法的引入，静态检测的概念被扩展为：

1)静态分析

对被测程序源代码进行扫描，从语法、语义上理解程序行为，直接分析被测程序特征，寻找可能导致错误的异常。

2)程序验证

通过对程序代码的形式化抽象，使用形式化验证技术证明程序是否符合特定安全规则，从而判断程序是否存在安全漏洞。

本文总结了目前常见的检测技术和手段，对静态检测可检测的漏洞类型进行了分类，描述和比较了各种技术，并展望了静态检测技术的前景。

2 检测漏洞的分类

静态检测关注程序内部特征，其技术特点与所检测漏洞的特征密切相关。因此，本节首先讨论静态检测的漏洞类型。

软件安全漏洞的分类方法很多。按已有分类方法，漏洞的区分较为细致，大部分现有静态检测技术覆盖的漏洞类型都很零散，很难在漏洞类型上发现其共性。

为便于比较，本文将漏洞分为：

(1)内存安全相关漏洞

关注数据流上的错误，通常由某些不正确的内存存储状态或使用情况导致。

(2)内存相关的安全漏洞

涉及数据和类型的正确性，因此，对于这一类漏洞的检测关键在于存储空间的建模。

时序安全相关漏洞关注控制流上的错误，往往由某些安全相关行为之间的不正确执行顺序导致。时序相关的安全漏洞涉及程序行为的时间关系，因此，只有偏序特性的方法才能检测。

根据这种划分，由于静态检测方法的技术特征不同和对漏洞的理解不同，某些方法只进行一类漏洞的检测，而有些方法则可以检测两类漏洞。

基金项目：国家自然科学基金资助项目(60473053)；国家“863”计划基金资助项目(2006AA01Z432)；江苏省自然科学基金资助项目(BK2005074)
作者简介：张 林(1983－)，男，硕士研究生，主研方向：信息安全；曾庆凯，教授
收稿日期：2007-08-28 **E-mail：**nju_Ares@hotmail.com

3 静态检测技术

从早期的缓冲区溢出检测开始，十几年来出现了各种检测技术，以下分程序静态分析和程序抽象验证两类来介绍。

3.1 静态分析

静态分析方法直接扫描程序代码，提取程序关键语法，解释其语义，理解程序行为，根据预先设定的漏洞特征、安全规则等检测漏洞。

(1)词法分析

词法分析是最早出现的静态分析技术，它仅仅进行语法上的检查。词法分析把程序划分为一个个片断，再把每个片断与一个“嫌疑数据库”进行比较，如果属于嫌疑，则进一步实行启发式判断。词法分析可以检查的漏洞较少，往往只是一些已知的固定漏洞代码，漏报率相当高。

(2)规则检查

程序本身的安全性可由安全规则描述。程序本身存在一些编程规则，即一些通用的安全规则，也称之为漏洞模式，比如程序在 root 权限下要避免 exec 调用。规则检查方法将这些规则以特定语法描述，由规则处理器接收，并将其转换为分析器能够接受的内部表示，然后再将程序行为进行比对、检测。

(3)类型推导

自动推导程序中变量和函数的类型，来判断变量和函数的访问是否符合类型规则。静态漏洞检测的类型推导由定型断言、推导规则和检查规则 3 个部分组成。定型断言定义变量的初始类型，推导规则提供了推论系统的规则集合，检查规则用于判定推论结果是否为“良行为”。

基于类型推导的静态分析方法适用于控制流无关的分析。但对于控制流相关的特性则需要引入类型限定词和子类型[4]的概念来扩展源语言的类型系统，使得新类型系统在源语言的数据类型上加以扩展并表示出类型之间的关系。类型限定词将变量类型表示的值的集合划分为几个互不相交的子集(子类型)，表示不同安全级别。子类型变量可在任何时候替换父类型变量，反之不可。利用类型限定关系可检测出越权限的访问错误。

3.2 程序验证

程序验证方法通过抽象程序得到形式化程序或模型，然后使用形式化验证技术进行检验证明，通过验证正确性的方式来检测漏洞[5]。

(1)模型检测

模型检测对有限状态的程序构造状态机或有向图等抽象模型，再对模型进行遍历以验证系统特性。一般有 2 种验证方式：

1)符号化方法将抽象模型中的状态转换为语法树描述的逻辑公式，然后判定公式是否可满足。

2)模型转换成自动机，并将需要检查的安全时序属性转换为等价自动机，再将这两个自动机取补，构成一个新的自动机，判定问题就变成检查这个新自动机能接受的语言是否为空。

模型检测需要列举所有可能状态，由于软件本身的高复杂度，对所有程序点进行建模可能会使模型规模庞大，因此一般只针对程序中某一方面属性构造抽象模型。近期出现的一种模型检测方法通过对内存状态的建模，从而使原先主要检测时序相关漏洞的模型检测方法可对内存相关漏洞进行相关检测。

(2)定理证明

定理证明比模型检测的形式化方法更加严格，用各种判定过程来验证程序抽象公式是否为真。判别的方法取决于公式的形式，如不等式的合取：首先由合取式构造成一个图，合取式中每个条件对应于图中的一个节点，然后利用给出的等式将对应的顶点合并，在顶点合并的过程中对合取式中的不等式进行检查，如发现存在不成立，则该合取式不可满足。

(3)符号执行

符号执行的基本思想是将程序中变量的值逻辑转换成抽象符号，模拟路径敏感的程序控制流，通过约束求解的办法，检测是否有发生错误的可能。由程序执行前的条件P出发，在程序中某程序点，可推出约束条件$c_1 \wedge c_2 \wedge \cdots \wedge c_n$，因此在该点有$P \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_n \rightarrow R$这样的规约式，其中$R$是程序结束后要满足的条件。对该规约式的否命题求解，如有一组解满足，说明在这个程序点上存在一组变量状态，使运行程序的结果不符。

符号执行求解工具的约束条件集合及求解能力决定了其发现错误的能力。理论上很多约束问题在可接受时间级内是不可解的，因此，符号执行求解的方法只适用于某些特定问题求解上。文献[6]提出的缓冲区模型其实就是将符号模型局限于缓冲区数据上，对其进行程序模拟并在每一步试图求解约束(缓冲区访问长度小于缓冲区长度)，其效率在可接受时间内。

3.3 检测方法的比较

大多数静态检测方法往往不是独立使用的，目前使用的很多检测工具常常同时使用多种检测方法。例如，常说的“数据流分析”工具，其实是类型推导和规则检查两种技术的共同运用，同时伴有符号执行的某些特征。表 1、表 2 比较了主要静态检测方法适用的漏洞类型和优缺点。

表 1 程序分析

检测方法	检测漏洞	优点	缺点
词法分析	内存相关	效率高	分析不精确，漏洞覆盖有限
类型推导	大部分内存相关；少量时序相关	能处理大规模程序，效率高	可检查漏洞有限，引入安全属性需重新定义类型
规则检查	内存相关，时序相关	能根据不同规则对不同系统进行分析，大规模程序检测高效	受规则描述机制局限，只能分析特定类型漏洞，扩展性差

表 2 程序验证

检测方法	检测漏洞	优点	缺点
模型检测	大部分时序相关；少量内存相关	能够严格检测程序时序上的漏洞，通过模型将漏洞放大	可能造成状态空间爆炸
定理证明	内存相关，时序相关	使用严格的推理证明控制检测的进行，误报率低	对某些域上的公式推理缺乏适用性，对新漏洞扩展性不高
符号执行	内存相关	精确地静态模拟程序执行，能发现程序中细微的逻辑错误	程序执行的可能路径随着程序规模增大呈指数级增长

程序验证比程序分析的理论基础更加严格，但其运行成本也更高。在程序全局性的检测上，程序验证的效果更加突出，而在程序局部性的检测上，程序分析更为高效。

在程序分析中，词法分析关注程序表面特征，过于简单；

类型推导和规则检查需要人工进行辅助定义类型和规则，扩展性自动性差，可检查漏洞也有限，但检测效率更高。

程序验证方面，模型检测的实用性已经得到实践证明，但其时序特性决定了其漏洞类型的局限性；定理证明需要使用者具有良好的理论素质，专业性较强，目前还没有较广泛的实现，效率也有待改进；符号执行将程序验证的严谨证明和程序分析的模拟扫描结合起来，但同样具有验证复杂、效率不高的缺点。

静态检测只能检测已知的漏洞类型。由于缺乏通用的漏洞描述机制，因此：

(1)对未知的漏洞，无法利用已知的漏洞特征对其进行规范描述。

(2)对于已知的漏洞，目前也很难有一种检测技术可以达到完全的覆盖率。

静态检测另外一方面的缺点是性能不足。静态检测的精确度同运行时间和空间消耗成正比，因此，提高检测质量的同时会增加运行成本。

静态检测技术衡量的 2 个重要指标为：

(1)漏报率(false negative rate)；

(2)误报率(false positive rate)。

降低其中之一的同时往往会造成另外一个指标的增高。

4 实现与趋势

从 2000 年开始，静态检测工具大量出现。在此之前，曾经出现了 Lint(1978)、ESC(1995)和 LClint(1994)，这些为后来的大部分工具起了奠基作用，但其本身并没有太多的实际使用价值。

(1)Lint 提供了最初的规则检查思想；

(2)LClint 在其上加以了改进，但需要用户手动加入注释以描述意图；

(3)ESC 则提供了抽象数据流符号执行的早期思想。

2000 年，出现了词法分析工具 ITS4、符号执行的工具 BOON 和 PREfix、一个简单的模型检测工具 Bandera。此时的符号执行仅仅只是简单地抽象数据，而模型检测 Bandera 也只是针对 Java 语言建立简单的有穷状态模型。

2001 年~2002 年，静态检测工具迅速发展。词法分析出现了分析更加准确的 RATS，符号执行方面则出现了抽象更加完备的 Mjolnir，在模型检测方面出现了 MOPS 和 SLAM 等经典工具，规则检查方面也出现了更为自动化的 MC 和 Splint。

同时还出现了定理证明工具 Eau Claire 和几个类型推导工具 CQual、CCured、ESP。静态检测主要技术在这两年都已经具有了雏形。

(1)2003 年，出现了 BLAST 和 RacerX，BLAST 在模型检测方面进行了优化，而 RacerX 为竞争条件漏洞检测提供了更好的方法。

(2)2005 年出现了 STLlint 和 Check’n’Crash，再次推动了符号执行技术的发展

(3)2005 年出现了定理证明工具 Cogent。

按所述技术分类对常见静态检测工具进行了归类，对应关系见表 3。

表 3 静态检测工具使用技术

静态检测技术	主要静态检测工具
词法分析	ITS4, Rats
类型推导	CCured, ESP, CQual
规则检查	MC, LCLint, Lint, RacerX, Splint
模型检测	Bandera, BLAST, MOPS, SLAM
定理证明	Cogent, EauClaire
符号执行	Mjolnir, ESC, C’n’C, BOON, STLlint, PREfix

静态检测技术目前的发展趋势是将静态检测的各种技术进行结合，以提高性能。

结合方式有以下几种方法：

(1)提供一个框架，使用不同检测技术对程序进行检测，获取大量检测结果之后进行分析。从误报率来说，使用多种技术的检测结果的交集来进行漏洞判断决策，可减少误报率；从漏报率来说，对于同一种漏洞，使用多种检测技术的结果的并集可以减少漏报率。

(2)直接在检测技术上结合，通过技术的结合来得到新的方法，如在符号执行的过程中加入类型推导的技术，在变量模拟执行的过程中增加其类型特征的推导，可获取更高的漏洞检测率。

除了多种静态检测技术的结合，还出现了动态和静态检测相结合的方法。SD(Static—Dynamic)方法先使用静态方法对程序进行分析得到程序内部特征，从而筛选测试数据集以指导下一轮的动态测试。DSD 方法在第 1 轮静态测试前进行一次动态测试，以排除某些不可能达到的输入情况，简化静态检测的复杂度。

5 结束语

静态检测技术试图理解程序，从检测技术本身来说，其具有一定优势，但是由于软件本身的复杂性，也会导致其检测过程的复杂度增加，增加检测精确度的同时往往伴随着空间和时间消耗的增加。

静态检测目前尚处在起步阶段，尽管已出现大量工具，但实用性不够，还需要更加深入的研究。鉴于静态检测的优势和作用，未来这一领域仍是值得探索的方向。

参考文献

[1] Weber S, Karger P A, Paradkar A. A Software Flaw Taxonomy: Aiming Tools at Security[C]//Proc. of ACM Software Engineering for Secure Systems——Building Trustworthy Applications. Louis, Missouri, USA: [s. n.], 2005.

[2] Landwehr C E. Formal Models for Computer Security[J]. ACM Computing Surveys, 1981, 13(3): 247-278.

[3] Xia Yiming. Security Vulnerability Detection Study Based on Static Analysis[J]. Computer Science, 2006, 33(10): 279-283.

[4] Foster J S, Fghndrich M, Aiken A. A Theory of Type Qualifiers[J]. ACM SIGPLAN Notices, 1999, 34(5): 192-203.

[5] Kurshan R P. Program Verification[J]. Notices of the American Mathematical Society, 2000, 47(5): 534-545.

[6] Wagner D. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities[C]//Proc. of the 7th Network and Distributed System Security Symposium. San Diego, USA,: [s. n.], 2000.