DOI:10.16511/j.cnki.qhdxxb.2012.10.001

ISSN 1000-0054 清华大学学报（自然科学版）2012 年 第 52 卷 第 10 期 1/29
CN 11-2223/N J Tsinghua Univ (Sci & Tech), 2012, Vol. 52, No. 10 1309-1319

软件漏洞分析技术进展

吴世忠， 郭 涛， 董国伟， 王嘉捷

（中国信息安全测评中心，北京 100085）

摘 要：软件漏洞是信息安全问题的根源之一，其造成的危害越来越大，因此软件漏洞分析逐渐成为信息安全理论研究和实践工作中的一个热门领域。该文首先定义了软件漏洞和软件漏洞分析技术，在此基础上，提出了软件漏洞分析技术体系，并对现有技术进行了分类和对比，归纳出了该领域的科学问题、技术难题和工程问题，最后展望了软件漏洞分析技术的未来发展。

关键词：信息安全；软件漏洞；漏洞分析

中图分类号：TP 309.2 **文献标志码**：A

文章编号：1000-0054(2012)10-0001-11

Software vulnerability analyses: A road map

WU Shizhong, GUO Tao, DONG Guowei, WANG Jiajie

(China Information Technology Security Evaluation Center, Beijing 100085, China)

Abstract: Software vulnerability is one cause of information security incidents which is becoming more and more damaging. Therefore, software vulnerability analyses are important in the theory and practice of information security. This paper describes software vulnerabilities and software vulnerability analysis methods, and then presents a software vulnerability analysis system. Various methods are compared to identify technical and engineering problems and future trends in software vulnerability analyses.

Key words: information security; software vulnerability; vulnerability analyses

随着全球信息化的迅猛发展，计算机软件已成为世界经济、科技、军事和社会发展的重要引擎。与此同时，软件的安全问题也日益突出。软件漏洞是安全问题的根源之一。随着互联网和软件技术的不断发展，软件漏洞的数量越来越多，造成的危害也越来越大，由其引发的信息窃取、资源被控、系统崩溃等问题会对国民经济、社会稳定等产生重大威胁。因此，对软件漏洞的研究日益受到重视[1]。

漏洞的普遍性及其后果的严重性促使研究人员将更多注意力集中于漏洞相关技术的研究上，包括漏洞检测（发现/挖掘）、漏洞特性分析、漏洞定位、漏洞利用、漏洞消控等。这些研究的开展在很大程度上促进了软件安全。

长期以来，对于软件漏洞尚没有统一的定义。经过广泛地研究和总结，本文将软件生命周期中涉及安全的设计错误（error）[2]、编码缺陷（defect）[2]和运行故障（fault）[2]统称为软件漏洞（vulnerability）；将发现和定位软件漏洞及对其原理进行分析的相关技术统称为软件漏洞分析技术。

基于此，本文对已有的软件漏洞分析技术进行综述。首先，根据软件生命周期和分析对象，将现有的分析技术划分为软件架构安全性分析技术、代码静态分析技术、代码动态分析技术、动静结合的分析技术和漏洞定位技术等 5 类，并总结出软件漏洞分析技术体系，该体系明确了各类技术之间的关系，给出了漏洞发现、定位和原理分析在整个漏洞分析技术框架中的准确位置。随后，对于每类技术，给出相应的原理模型，并从定义、操作步骤、突出优势、适用范围、局限性、工具实现等方面详细介绍所包含的具体技术，在此基础上，对它们进行对比说明。最后，归纳软件漏洞分析领域的科学问题、技术难题和工程问题，进而对未来的发展方向进行展望。

1 软件漏洞分析技术

软件漏洞分析是信息安全和软件质量保障的重要手段，分析工作的开展通常与软件生命周期紧密地结合在一起：1）在设计阶段，进行软件架构的安全性分析，了解软件架构中存在的安全威胁，由此发现设计错误（error），以尽早消除安全隐患；2）在

收稿日期：2012-08-27

基金项目：国家自然科学基金资助项目（90818021，61100047）

作者简介：吴世忠（1962—），男（苗），湖南，研究员。

E-mail: guotao@itsec.gov.cn

开发阶段,进行源代码的安全性分析,了解代码自身存在的固有问题,由此发现编码缺陷(defect)以弥补编码的不足;3)在测试阶段,进行可执行代码的安全性分析,了解软件在实际运行中可能出现的安全问题,由此发现运行故障(fault),以采取相应措施补救。

漏洞分析技术多种多样,其间的界限也不清晰。

1.1 漏洞分析技术分类

综合考虑分析对象、漏洞形态等因素,参考已有软件分析分类方法[3],将软件漏洞分析划分为软件架构分析技术、代码静态分析技术、代码动态分析技术、动静结合的分析技术和漏洞定位技术等5类。图1展示了软件漏洞分析的技术体系。其中,架构分析技术是对软件的架构设计进行分析,发现违反安全属性的设计错误后反馈给设计人员进行修改;代码静态分析技术和代码动态分析技术的分析对象是源代码或可执行代码,分析过程中需要辅以相应的分析规则,输出可能的安全漏洞;有时为了提高效率和准确度,需将两者结合使用;漏洞定位技术主要对是漏洞进行追踪定位等分析,由此确定漏洞的位置、可利用性等属性。上述各类技术构成一个完整的体系,能针对软件的不同形态完成漏洞的发现和分析工作。

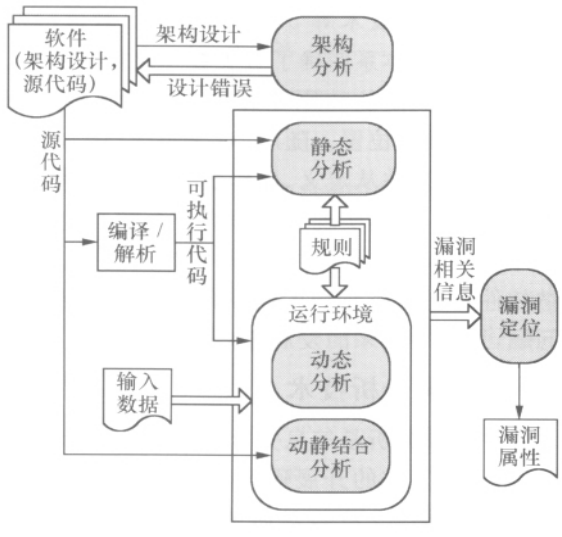

图1 软件漏洞分析技术体系

1.2 软件架构分析

统计发现,软件中50%~75%的问题是在设计阶段引入的[4],且修复成本会随发现时间的推迟而增长。因此,在设计阶段进行软件架构分析对软件安全保障起着决定性作用,软件架构分析也提供了从更高、更抽象的层次保障软件安全性的方法。

图2是软件架构分析原理图。分析时首先要对架构进行建模,同时还要对软件的安全需求或安全机制进行描述,然后检查架构模型是否满足安全需求,如不满足,需要根据相应信息重新设计架构,如此反复直至满足所有安全需求。通常,为了便于(自动化)检查,为软件架构和安全需求建模/描述时使用相同的标准进行构造,即得到的模型形式相同。

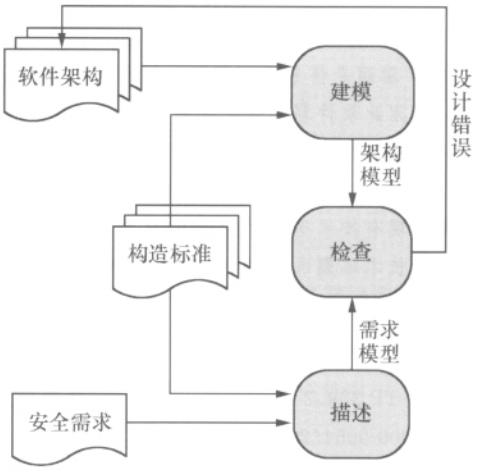

图2 软件架构分析原理

目前,国内外关于软件架构安全性分析的理论和应用研究都还处在探索和发展阶段,主要可以分为形式化分析和工程化分析2大类。前者主要使用形式化方法描述软件架构和安全需求,因此最终的分析结果精确、可量化,且自动化程度高,但实用性较差;后者从攻击者的角度考虑软件面临的安全问题,实用性强,但自动化程度较低。具体而言:

1)形式化分析

UMLsec方法通过扩展标准的统一建模语言(UML)来描述软件的架构和安全需求,并将它们转化为一阶逻辑公式,最终自动验证软件的架构是否满足安全需求[5]。UMLsec能够进行自动化的分析,具有一定的实际应用价值;但建模粒度相对较粗。

软件架构模型法(SAM)使用谓词/迁移网(PrT)描述架构基本元素,使用一阶时序逻辑描述安全约束,并通过一系列形式化验证和分析方法判定基本元素是否满足安全约束[6]。SAM使用模型检查、约束求解、定理证明、可达性分析等方法,能够实现自动化,架构描述能力强;但需要复杂的形式化表示,实用性较差。

DTMC方法使用离散时间Markov链描述软件

架构的状态空间以反映任意时刻各组件的运行状况，而后人工输入每个组件的脆弱性指数，并以此计算出软件的安全性及影响架构安全的瓶颈[7]。DT-MC 支持层次式建模和多种分析及预测过程；但需要人工干预，安全描述能力较弱，且实用性不强。

ACME 方法使用组件系统的架构描述语言描述软件架构的基本元素，同时还描述基本元素的性质以体现安全需求，最终通过体系结构模拟法分析架构的安全性[8]。ACME 的安全描述能力较强，具有一定的实用价值；但分析时需要人工干预。

2）工程化分析

场景分析法使用场景描述与软件架构的静态结构和动态行为相关的安全属性，并采用评审会议的方式分析架构安全性[9]。场景分析可在上下文环境中描述复杂的安全属性，不依赖于特定的架构描述语言；但需要人工评审，自动化程度低。

错误用例分析法通过检查软件架构对每个错误用例（用户在与软件交互过程中，对其他用户、软件等造成损失的一系列行为）如何反应来判断架构是否满足安全需求[10-11]。错误用例分析法提高了安全问题的可视化，并且不依赖于某种特定的架构描述语言；但需要人工评审，不能进行自动化分析。

威胁建模法[12]通过建立分层数据流图、标识软件的入口点和信任边界来描述软件架构，并通过建立威胁模型（STRIDE）来展现欺骗、篡改、否认、信息泄露、拒绝服务和特权提升等 6 类威胁，最终使用 SDL TM[13]工具辅助完成分析。威胁建模法可以借助工具自动完成部分分析工作，实用性较强；但对于安全属性的描述能力较弱。

1.3 代码静态分析

代码静态分析是指在不运行软件的前提下进行的分析过程[3]。静态分析的对象通常是源代码，也可以是可执行代码，但针对源代码分析时会获得更多的语义信息，更便于分析。使用静态分析方法，可以比较全面地考虑执行路径的信息，因此能够发现更多的漏洞，提高命中率。

图 3 给出了代码静态分析的原理。可以看出，整个过程包括如下环节：软件模型构造、漏洞模式提取、基于软件模型和漏洞模式的模式匹配。其中，匹配技术一般是在辅助分析技术的支持下采用数据流分析、符号执行、模型检查等完成的；软件模型的构造应当与所使用的分析技术相结合，一般是从软件的设计、实现中构造出来的，如语法树（AST）、控制流图（CFG）、形式化规约等，通常被统称为中间表示（IR）[14-18]。在建模之前，往往需要对代码进行预处理（词法语法分析等）；漏洞模式提取过程也是分析标准的构造过程，根据历史漏洞的类型和特性，构建相应的漏洞模式或分析规则。

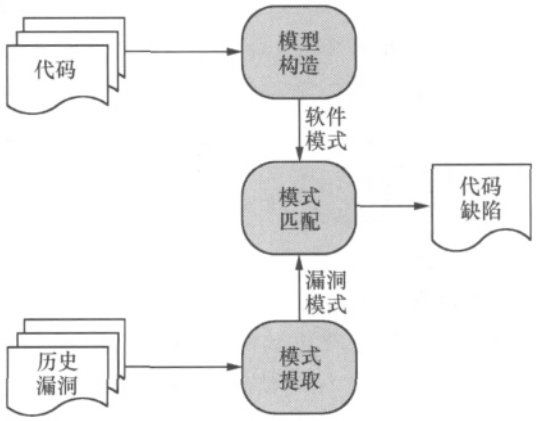

图 3 代码静态分析原理

经过几十年的发展，各类静态分析技术已比较成熟。目前比较常见的静态分析技术包括词法分析、数据流分析、符号执行、模型检查、定理证明、污点传播等，具体而言：

1）词法分析

词法分析只对代码进行基于文本或 Token 流的对比，以查找危险的函数或 API[19]。词法分析能够快速地发现软件中的危险函数，检测效率较高；但是由于其不进行语义层面的分析，因此无法发现深层次的安全问题，且漏报率和误报率较大。使用该技术的源代码漏洞扫描工具包括 Checkmarx[19]和 ITS4[20]等。

2）数据流分析

数据流分析通过确定程序某点上变量的定义和取值情况来分析潜在危险点[21-23]。数据流分析是许多代码静态检测技术的基础，在进行数据流分析时，通常将代码构造为抽象语法树（AST）和程序控制流图（CFG）等模型，而后通过代数方法计算变量的定义和使用，描述程序运行时行为，进而根据相应规则发现漏洞。数据流分析具有较强的分析能力，适合检查需要考虑控制流信息而且变量属性之间的操作较为简单的问题，如内存访问越界、常数传播等；但该技术分析速度低、过程间分析较为复杂、容易出错且效率低下。使用该技术的漏洞分析工具包括 Coverity[24]、Klocwork[25]、JLint[26]等。

3）符号执行

符号执行通过模拟程序执行发现安全问

题[27-28],通常与约束求解技术结合使用[29]。进行符号执行时,通常将代码构造为控制流图(CFG)或函数调用图(CG)以进行过程内或过程间分析,而后使用符号值表示程序变量值,并模拟程序的执行来查找满足漏洞检测规则的情况,从而发现漏洞。符号执行能够发现程序中细微的逻辑错误,是一种高效的静态漏洞分析技术;但是程序执行的可能路径随着程序规模的增大呈指数级增长,即出现"空间爆炸"问题,并且发现问题的能力很大程度上取决于求解工具的求解能力。使用该技术的漏洞分析工具包括 PREfix[30]、SMART[31]和 SAGE[32]等。

4) 模型检查

模型检查基于状态迁移系统来判断程序的安全性质[33-35]。模型检查将软件构造为状态机或者有向图等抽象模型,并使用模态/时序逻辑公式等形式化的表达式来描述安全属性,而后对模型进行遍历以验证软件的这些安全属性是否满足。模型检查对于路径和状态的分析结果准确性较高,分析过程可以实现全自动化;但是由于穷举了所有可能状态,所以增加了额外的开销,特别是当数据密集度较大时,分析难度增大;此外,对时序、路径等属性,在边界处的近似处理难度也较大。使用该技术的漏洞分析工具包括 MOPS[34]、SLAM[36]、Java PathFinder[37]等。

5) 定理证明

定理证明将待验证问题转化为数学上的定理证明问题来判定程序是否满足特定安全属性[38],其中的 Boole 可满足性分析是目前研究的热点[39-40]。定理证明方法将程序转换为逻辑公式,然后使用公理和规则证明程序是否为一个合法的定理,从而发现其中的漏洞。定理证明使用严格的推理证明控制分析过程,是众多静态分析技术中最准确的,误报率较低;但是该技术需要大量的人工干预,自动化程度较低,对新漏洞的扩展性不高,难以广泛应用于大型程序中。使用该技术的漏洞分析工具主要包括 Saturn[39]、ESC/Java[41]等。

6) 污点传播

污点传播(tainted propagation)分析通过静态跟踪不可信的输入数据来发现安全漏洞[42-44]。污点传播通过对不信任的输入数据进行标记,静态跟踪程序运行过程中污点数据的传播路径,发现使用污点数据的不安全方式,进而分析出由于敏感数据(如字符串参数)被改写而引发的输入验证类漏洞,如跨站脚本、SQL 注入等。污点传播分析能够模拟攻击的整个过程;但它只适用于输入验证类漏洞的分析。Pixy 是使用该技术的漏洞分析工具的典型代表[43-44]。此外,Fortify SCA 也具有多维污点传播分析能力[45]。

1.4 代码动态分析

动态分析是通过运行具体程序并获取程序的输出或者内部状态等信息来验证或者发现软件性质的过程[2,46],分析对象是可执行代码。使用动态方法分析漏洞,由于获取了具体的运行信息,因此分析出的漏洞一般更为准确,误报率较低。

代码动态分析的本质是使用构造的特定输入运行软件[47],出现故障(通常是崩溃)时即表示非法数据触发了一个可疑漏洞,图 4 说明了其原理。其中,软件运行环境用于控制待分析软件的运行并观察运行状态,包括软件的启动、停止、输出获取等;环境控制用于监控软件的运行;最终通过观察输出来确定相应输入是否触发了可能的漏洞,当输入数据较多时,需要对所有输出进行分析过滤,只保留与漏洞相关的信息。

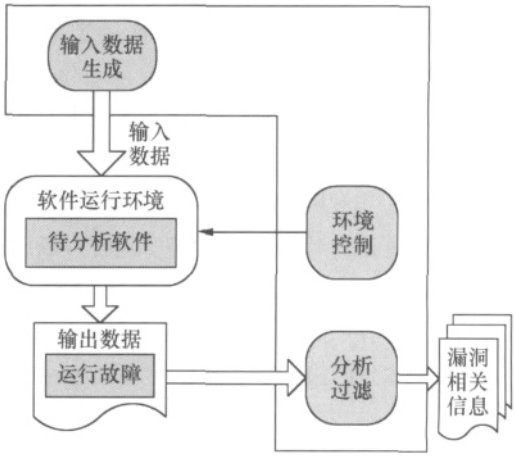

图 4 代码动态分析原理

目前,动态漏洞分析技术主要包括模糊测试、智能模糊测试和动态污点分析等,具体而言:

1) 模糊测试

模糊测试(fuzzing)通过提供非正常输入并监测运行故障来发现软件漏洞,通常包括识别目标(识别待分析程序的结构、历史漏洞等)、识别输入(确定输入向量或格式)、生成模糊测试数据、执行测试数据、监视异常、确定漏洞可利用性等 6 个阶段[48]。模糊测试是一种有效的动态漏洞分析技术,使用该项技术已经发现了大量的未知漏洞[49-50];但是测试数据生成的随机性导致了分析效率低下,代码覆盖的不充分性导致了漏报率偏高,测试数据的相互独

立性导致了复杂漏洞难以发现。比较著名的模糊测试工具包括 SPIKE[49]、Peach[50]等。

2）智能模糊测试

智能模糊测试(smart fuzzing)通常首先对待分析软件的代码进行静态分析，获取部分结构特征或语义信息，然后有针对性地设计测试数据，以此来辅助模糊测试的开展[51]。智能模糊测试更具针对性，因此提高了分析的准确性；但是静态分析的引入增加了分析的强度和难度，所以使用该技术时需要平衡效率和准确率之间的关系。TaintScope 是一款优良的智能模糊测试工具[51]。

3）动态污点分析

动态污点分析(dynamic taint analysis)能够在机器指令层面对运行软件中的不可信数据进行信息流跟踪，从而发现不安全行为[52]。分析时，首先动态地将来自污点源(taint source)的数据标识为污点数据，而后通过分析软件执行的指令和操作数，按照某种传播策略跟踪和记录污点数据的传播路径；当污点数据传播到敏感操作点(sink 点)时，即按预定的策略触发相应操作。动态污点分析已被广泛应用于漏洞分析、恶意代码分析、隐私泄露分析等领域；但通常需要基于可执行代码分析技术，而可执行代码的分析面临着应用语义和类型信息缺失的问题。BitBlaze 是动态污点分析方面较为突出的工具[52]。

1.5 动静结合分析

静态分析比较全面地考虑执行路径，漏报率比动态分析低；但动态分析由于获取具体的运行信息，因此报告的漏洞更为准确，误报率较低[3]。为发挥各自的优点，有时也将这 2 种技术结合起来使用。

目前的动静结合漏洞分析技术主要包括 3 类：1）对源代码进行静态分析，发现可能的漏洞，然后构造输入数据使用动态分析验证其真实性[53-56]；2）对可执行代码进行反汇编，利用对汇编代码静态分析获取的信息指导动态漏洞分析[57]；3）执行待分析软件并搜集运行信息，在汇编代码级别上指导静态漏洞分析[58-59]。

第一类技术目前使用最为广泛，但是它只考虑到了静态分析对动态分析的指导作用，而忽略了动态分析过程中产生的运行信息的价值；第二类技术是对动态污点分析技术的改进，但是对可执行代码反汇编得到的汇编代码缺少语义信息，并且对可执行代码运行过程的跟踪难度较大；第三类技术考虑到了运行信息，较为新颖，但仍处于探索阶段。

1.6 漏洞定位

漏洞定位是指在代码中确定漏洞发生的位置，了解产生机理，准确评估潜在利用方式和风险等级的过程。前述的各类技术其实也包含着漏洞定位的分析：架构分析时，需要对发现的设计错误溯源，分析导致问题的软件元素和不合理结构；静态分析时，发现的可能漏洞需要进一步验证，并且现有的静态漏洞分析工具都可以生成辅助分析的图形或触发路径，便于漏洞的定位、机理分析和溯源[24-25,45]；动态分析时，需要从大量输出数据和软件运行状态信息中分析确定漏洞，如图 4 中的分析过滤。

目前，漏洞定位技术主要在可执行代码中进行，包括动态调试、逆向分析和指令追踪等技术。

1）动态调试

动态调试通常用来定位漏洞和发现漏洞利用方法，分析对象是可执行代码。动态调试需要在调试器中运行软件，并观察软件的状态、内存使用情况和寄存器值等内容来实现。调试时通过给定的输入，跟踪分析运行时堆栈、寄存器状态和函数调用路径来发现触发漏洞的根源，定位漏洞。OllyDbg 是一款知名的动态调试工具[60]。

2）逆向分析

逆向分析时首先对可执行代码进行反汇编，获得汇编代码，然后通过扫描汇编代码识别出可疑的代码序列，进而定位出漏洞。逆向分析工具的典型代表是 IDA Pro[61]。

3）指令追踪

指令追踪也是一种逆向分析技术，能够迅速定位漏洞。使用指令追踪时，首先需要正常运行软件，记录所有执行过的指令序列，然后触发漏洞，记录漏洞触发状态下执行的指令序列，最终比较 2 轮执行的指令，重点逆向分析 2 次执行中表现不同的代码区，并动态调试和跟踪这部分代码，从而定位漏洞函数。基于动态可执行代码插桩工具 PIN[62]、Valgrind[63]等进行指令追踪是目前较为流行的方法。

1.7 分析技术评价

研究发现，目前还没有对软件漏洞分析技术进行系统归纳总结的工作。为了更加清晰地认识上述各类技术，本文对它们从分析对象(输入)、分析结果(输出)、软件模型、具体分析技术、应用状况和优缺点等方面进行详细对比说明，如表 1 所示，其中“—”表示相应技术在实施过程中不涉及该项内容。

表1 软件漏洞分析技术对比

分类	分析对象	结果	软件模型	具体技术	应用状况	优点	缺点
架构分析	架构设计	设计错误	形式化模型或特殊形式的模型	UMLsec、DT-MC、场景分析、威胁建模等	处于探索和发展阶段	考虑软件整体安全性、在软件设计阶段进行、抽象层次高	缺少实用且自动化程度高的技术
静态分析	源代码或可执行代码	代码缺陷	AST、CFG、Token、形式化模型等	词法分析、数据流分析、符号执行、模型检查、定理证明等	大部分技术已比较成熟,处于瓶颈期	代码覆盖率高、能够分析出隐藏较深的漏洞、漏报率较低	需要大量人工辅助、技术难度大、对先验知识(历史漏洞)的依赖性较大、误报率较高
动态分析	可执行代码	运行故障	—	模糊测试、智能模糊测试、动态污点分析等	智能模糊测试和动态污点分析是目前研究热点	实施简单、能够获得具体的运行信息、分析结果可重现、误报率较低	代码覆盖率低且难以度量、对于复杂漏洞的分析能力低、漏报率较高
动静结合	源代码或可执行代码	运行故障	AST、CFG、汇编语言等	静态分析技术和动态分析技术结合使用	处于发展阶段,是目前的研究热点	结合了静态分析和动态分析的优点	对可执行代码的静态分析缺少语义信息、运行信息对静态分析的指导不足等
漏洞定位	可执行代码	漏洞的位置可利用性等属性	汇编语言	动态调试、逆向分析、指令追踪等	比较成熟,指令追踪是目前研究热点	能够对漏洞的攻击原理进行深入的分析,确定漏洞的本质	只能在工具辅助下人工完成,缺少自动化、规模化分析技术

从表中可以看出:1)综合使用这些技术,可以对生命周期各个阶段中不同形态的软件进行系统地漏洞分析,从而达到软件安全保障的目的。2)对于软件架构安全性分析技术的研究,目前仍处于探索和发展阶段。由于此类技术在软件开发的早期,从较高、较抽象的层次发现安全问题,能够减少危害损失和修复成本,所以必将成为今后漏洞分析领域研究的热点,并且重点应放在研究实用且自动化程度高的分析技术上。3)对于代码漏洞分析技术的研究,无论是静态分析还是动态分析,经历了几十年的发展,均已比较成熟且进入了瓶颈期,单纯对静态分析技术或动态分析技术进行改进,其性能都已没有太大的提升空间。但是,由于2种技术的优缺点是互补的,因此动静结合的漏洞分析技术已经并将继续成为研究的热点,但需要解决可执行代码静态分析准确度、运行信息收集、测试数据自动化生成等难题。4)漏洞定位技术是了解漏洞本质和产生机理的关键,但目前需要大量人工参与,缺少自动化、规模化的分析技术。

2 软件漏洞分析中的关键性问题

虽然软件漏洞分析技术已取得了巨大的进展,但还存在着一些亟待解决的关键性问题。本节从科学研究、技术应用和工程实现等3个方面阐述目前软件漏洞分析领域中普遍存在的问题和难题。

2.1 科学问题

1)软件模型构建

软件漏洞分析的对象通常是源代码或可执行代码,分析时往往需要对它们建模,即将软件转化为中间表示(IR),而后在其上开展自动或半自动的分析。常见的软件模型包括树型结构和图型结构等。不同的模型适用的漏洞分析技术也不尽相同,例如,抽象语法树(AST)、控制流(CFG)等是早期漏洞分析中常用的模型,最近的研究尝试使用统一中间表示对软件的源代码和可执行代码建模[64-66]。然而,由于现有的软件建模方法大多源于程序编译优化技术,因此许多中间表示并不是专门针对漏洞分析的。所以,需要深入研究面向漏洞分析的软件建模方法,使模型能够全面反映软件的属性,尤其是安全特性,以提高漏洞分析的准确度。

2)漏洞模式提取

漏洞模式的提取是关系到漏洞分析效果的关键因素,常见的漏洞模式包括适合类型安全分析的类型约束模式[67]、适合危险函数调用分析的语法结构

模式[68]、适合污点传播分析的格(lattice)与不动点(fixpoint)模式[69]等。此外，在进行模型检查时，漏洞模式被描述为模态/时序逻辑公式[34]。但现有漏洞模式的描述粒度较粗，对漏洞性质尤其是其运行上下文的表述普遍不够全面，会引发大量误报。因此，需要进一步研究漏洞模式的提取方法，使漏洞模式能够准确反映漏洞的动静态属性，并指导漏洞发现规则的构造，进而在确保自动化分析的前提下提高分析的准确度。

3) 技术极限求解

与许多其他研究领域类似，软件漏洞分析也存在极限问题，即在可接受的时间空间成本范围内，能够发现和分析的漏洞种类及特性，漏洞分析的覆盖率、深度和精度[70]，以及能够通过漏洞分析检验的安全属性种类和特性等[71]。通过对现有漏洞分析技术进行归纳总结，找到技术极限与现有技术体系不足之间的关联关系，而后通过结合使用多种分析技术、规范待分析软件的架构、使用限制性较强的程序设计语言等途径，发现上述极限问题的近似解决方法，也是下一步需要深入研究的方向。

2.2 技术难题

1) 精确度判定

为了提高漏洞分析能力，需要研究更加精确的分析方法，例如，漏洞分析所依赖的基本分析技术按照分析精度从简单的语法匹配、流不敏感分析、流敏感分析，逐步上升至路径敏感的分析[72]，这些都在不断地提高着漏洞分析的准确度和深度。但是，由于核心算法的时间空间复杂度快速提升，不可避免地会导致计算资源消耗的增加，更为严重的是，这种增加往往是几何级的，例如，路径敏感分析时会面临路径爆炸问题[73]。因此，不断提高漏洞分析的精确判定程度，在合理的时间和资源条件下有效缓解漏洞分析精度和资源消耗之间的矛盾，是漏洞分析领域需要解决的技术难题。

2) 智能化提升

软件漏洞分析是一项对智力和技术水平要求都很高的工作。目前，漏洞分析领域涌现出许多方法、技术和工具，逐步摆脱了之前依靠手工经验和密集劳动的状况。但是，现有的漏洞分析技术智能水平不高，依然在很大程度上依赖于分析人员的漏洞先验知识[74]，例如，静态分析技术大多基于历史漏洞的特征[75]，而动态分析往往基于漏洞攻击与异常输入等知识[76]。但相应知识的总结和提取非常困难和耗时，漏洞分析智能化提升和先验知识自动获取方面的研究是消除漏洞分析领域瓶颈的重要途径。

2.3 工程问题

1) 大规模软件的漏洞分析

虽然软件的规模和复杂性不断增加，但现有的软件漏洞分析工具大多数仍基于单一引擎或单机进行集中分析，且分析速度不高。分布式并行化分析能够极大地提高漏洞分析效率，如现有的分布式并行化模糊测试工具、并行化过程间程序分析工具、并行化符合执行系统等[77]，但此类技术仍处于探索阶段，工程应用效果并不突出。因此今后需要研究分析算法的升级改进，设计面向漏洞分析的分布式并行化算法，以充分利用分布式系统和多核硬件架构的性能优势，并解决各算法模块间的运行时动态协同、大规模分析任务的合理分配和及时调整等问题，以提升对大型复杂软件的分析效率。

2) 新型平台上的漏洞分析

随着移动互联网技术的发展，以 Android、iOS 为代表的移动智能终端平台迅速普及，构建在其上的电子商务、网银支付、即时通讯、社交网络、影音互动、在线游戏等类型的应用软件也越来越多，并且已逐渐成为漏洞分析的主要对象。这些软件大多是基于 ARM 架构和 Dalvik 虚拟机等新型平台开发的，而当前主流漏洞分析工具主要面向 X86/X64 架构和 Windows/Linux 等通用平台上的软件，不能直接使用。因此，需要在移动智能操作系统[78-80]、云服务平台[81]等领域，探索新型平台相关的漏洞分析技术，并开发能够应用于这些平台的漏洞分析工具。

3) 安全测评中的漏洞分析

目前，中国的信息安全测评工作主要基于通用信息安全测评准则(CC)和 GB/T 18336，大多关注于系统功能的安全性和系统安全功能的实现等方面。但是，CC 等标准给出的针对系统功能安全的穿透性测试之类的方法不够系统化，具体的可操作性指导不多，而这些方法恰恰属于漏洞分析的范畴。因此，需要研究漏洞分析在安全测评中的作用和应用。例如，可以对多个安全需求相似的软件系统或同一个软件系统的不同变体，在给定的上下文环境中，利用漏洞分析技术发现它们中存在的漏洞的特征，并在此基础上比较它们

的安全性。这就需要深入研究基于漏洞分析的可量化的软件系统安全性度量标准[82],使其能够客观全面地反映系统的安全属性,进而更好地服务于安全测评工作。

3 未来展望

软件漏洞分析是一项极具挑战性的工作,技术难度也非常大,需要综合多个领域的技术才能有所突破创新。根据前面章节的总结,本节归纳出了该领域未来的发展方向。

3.1 探索前沿科学问题,推动漏洞分析技术的创新

积极探索科学问题,推进面向漏洞分析的软件建模和漏洞模式提取等方面的研究,探讨软件系统统一建模方法,构造对漏洞特征描述全面的漏洞模式,以指导漏洞发现规则的构造和漏洞分析自动化的开展。此外,深入研究漏洞分析技术体系的技术极限,找到技术极限与现有分析方法的不足之间的关联关系,为突破技术极限奠定基础,并积极推动软件漏洞分析技术的基础性创新。

3.2 研究关键技术问题,实现漏洞分析技术的突破

针对制约漏洞分析效率的关键性技术问题,切实推进提高漏洞分析效率和准确度的算法的研究,逐渐实现漏洞分析的精确判定,减少人工辅助分析的工作量。虽然难以完全解决分析精度和资源消耗之间的矛盾,但如果能将分析目标集中在安全敏感区域中,还是能够在不降低分析精度的前提下,依靠合理的计算资源获得较好的分析结果的。同时,重点开展提高漏洞分析智能化水平的研究,探索漏洞分析知识的自动化获取方法,有效缓解传统的漏洞分析技术对人工总结的漏洞先验知识的依赖。

3.3 解决现实工程问题,促进漏洞分析技术的应用

实现自动化、规模化地分析方法是当前漏洞分析领域面临的工程难题。可以在当前并行算法的基础上,积极研究分布式并行化漏洞分析技术,尽快形成规模化的工程应用和坚实的漏洞分析能力。此外,为了使漏洞分析技术能够适用于信息技术的快速发展,需要有针对性地研究面向移动互联网、云计算等新型平台的漏洞分析技术,并实现相应工具。同时,还需要建立基于漏洞分析的安全度量体系,推动安全测评工作向更高更准的方向发展。

参考文献 (References)

[1] 吴世忠. 信息安全漏洞分析回顾与展望[J]. 清华大学学报:自然科学版, 2009, **49**(S2): 2065-2072.
WU Shizhong. Review and outlook of information security vulnerability analysis [J]. *Journal of Tsinghua University: Science and Technology*, 2009, **49**(S2): 2065-2072. (in Chinese)

[2] 陆民燕. 软件可靠性工程[M]. 北京:国防工业出版社, 2011.
LU Minyan. Software Reliability Engineering [M]. Beijing: National Defense Industry Press, 2011. (in Chinese)

[3] 梅宏, 王千祥, 张路, 等. 软件分析技术进展. 计算机学报[J]. 2009, **32**(9): 1697-1710.
MEI Hong, WANG Qianxiang, ZHANG Lu, et al. Software analysis: A road map [J]. *Chinese Journal of Computers*, 2009, **32**(9): 1697-1710. (in Chinese)

[4] Pressman R. Software Engineering [M]. A Practitioner's Approach. 4th edition. New York, USA: McGraw Hill, 1997.

[5] Jurjens J. UMLsec: Extending UML for secure systems development [C]// Proceedings of the 5th International Conference on The Unified Modeling Language. London, UK: Springer-Verlag, 2002: 412-425.

[6] DENG Yi, WANG Jiacun, Jeffrey P J. An approach for modeling and analysis of security system architectures [J]. *IEEE Transaction on Knowledge and Data Engineering*, 2003, **15**(5): 1099-1119.

[7] Sharma V, Kishor S, Trivedi. Architecture based analysis of performance, reliability and security of software system [C]// Proceedings of the 5th International Workshop on Software and Performance. New York, USA: ACM Press, 2005: 217-227.

[8] Garlan D, Schmerl B. Architecture-driven modeling and analysis [C]// Proceedings of the 54th International Conference on the Society for Modeling & Simulation International. Darlinghurst, Australia: Australian Computer Society, 2006: 3-17.

[9] Kazma R, Abowd G, Bass L. Scenario-based analysis of software architecture [J]. *IEEE Software*, 1996, **13**(6): 47-55.

[10] Pauli J, XU Dianxiang. Misuse case-based design and analysis of secure software architecture [C]// Proceedings of the International Conference on Information Technology: Coding and Computing. Piscataway, USA: IEEE Press, 2005: 210-225.

[11] McDermott J, Fox C. Using abuse case models for security requirements analysis [C]// Proceedings of the 15th Annual Computer Security Applications Conference. Piscataway, USA: IEEE Press, 1999: 112-124.

[12] Swiderrski F, Snyder W. Threat Modeling [M]. Seattle, USA: Microsoft Press, 2004.

[13] Microsoft Corporation. The Microsoft SDL threat modeling tool [Z/OL]. (2010-06-30), http://msdn.microsoft.com/en-us/security/dd206731.aspx.

[14] 吴世忠, 高新宇, 易锦. 基于 Petri 网的信息安全漏洞建模与分析[J]. 清华大学学报:自然科学版, 2010, **50**(S1): 1489-1495.

WU Shizhong, GAO Xinyu, YI Jin. Information security vulnerability modeling and analysis based on Petri networks [J]. *Journal of Tsinghua University: Science and Technology*, 2010, **50**(S1): 1489-1495. (in Chinese)

[15] 易锦，郭涛，黄永刚，等. 基于语言等价关系化简 Buchi 自动机的算法 [J]. 清华大学学报：自然科学版，2009，**49**(S2)：2181-2185.
YI Jin, GUO Tao, HUANG Yonggang et al. Optimization of Buchi automata by language equivalence relationships [J]. *Journal of Tsinghua University: Science and Technology*, 2009, **49**(S2): 2181-2185. (in Chinese)

[16] 熊浩，晏海华，郭涛，等. 代码相似性检测技术 [J]. 计算机科学，2010，**37**(08)：9-15.
XIONG Hao, YAN Haihua, GUO Tao et al. Code similarity detection: A survey [J]. *Computer Science*, 2010, **37**(08): 9-15. (in Chinese)

[17] 梁洪亮，陈政，张普含. ABAR：基于源代码的缺陷自动分析 [J]. 清华大学学报，2010，**50**(S1)：1597-1602.
LIANG Hongliang, CHEN Zheng, ZHANG Puhan. ABAR: Automatic bug analyzer based on source code [J]. *Journal of Tsinghua University: Science and Technology*, 2010, **50**(S1): 1597-1602. (in Chinese)

[18] 董国伟，郭涛，时志伟，等. Java 代码安全缺陷分析软件研究 [C]// 第四届信息安全漏洞分析与风险评估大会论文集. 北京：中国信息安全测评中心，2011：205-211.
DONG Guowei, GUO Tao, SHI Zhiwei et al. Study of security flaw analysis software for Java [C]// Proceedings of the 4th Conference on Vulnerability Analysis and Risk Assessment. Beijing: China Information Technology Security Evaluation Center, 2011: 205-211. (in Chinese)

[19] Checkmarx Corporation. Checkmarx CxSuite Enterprise. [Z/OL]. (2011-03-21), http://www.checkmarx.com/.

[20] Viega J, Bloch J T, Kohno Y. ITS4: A static vulnerability scanner for C and C++ code [C]// Proceedings of the 16th Annual Computer Security Applications Conference. New York, USA: ACM Press, 2000: 212-221.

[21] 许中兴. C 程序的静态分析 [D]. 北京：中国科学院软件研究所，2009.
XU Zhongxing. Static Analysis of C Programs [D]. Beijing: Institute of Software of Chinese Academy of Sciences, 2009. (in Chinese)

[22] Larochelle D. Statically detecting likely buffer overflow vulnerabilities [C]// Proceedings of the 10th USENIX Security Symposium. New York, USA: ACM Press, 2001: 139-148.

[23] XIE Yichen, Chou A, Engler D. ARCHER: Using symbolic, path-sensitive analysis to detect memory access errors [C]//Proceedings of the 9th European Software Engineering Conference Held Jointly with the 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA: ACM Press, 2003: 220-233.

[24] Coverity Corporation. Coverity Static Analysis [Z/OL]. (2011-04-25), http://www.coverity.com/.

[25] Nortel Networks. Klocwork [Z/OL]. (2011-02-12), http://www.klocwork.com/.

[26] Artho C, Biere A. Applying static analysis to large-scale [C]// Proceedings of the 13th Australian Conference on Software Engineering. Multi-Threaded Java Programs. Piscataway, USA: IEEE Press, 2001: 194-208.

[27] ZHANG Dazhi, LIU Donggang, WANG Wenhua, et al. Testing C programs for vulnerability using trace-based symbolic execution and satisfiability analysis [C]// Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'10). Piscataway, USA: IEEE Press, 2010: 321-338.

[28] Ganapathy V, Jha S, Chandler D. Buffer overrun detection using linear programming and static analysis [C]// Proceedings of the 10th ACM Conference on Computer and Communications Security. New York, USA: ACM Press, 2003: 109-120.

[29] Alexander A. Introduction to set constraint-based program analysis [J]. *Science Computer Program*. 1999, **35**(2): 79-111.

[30] Bush W, Pincus J, Sielaff D. A static analyzer for finding dynamic programming errors [J]. *Software Practice & Experience*, 2000, **30**(7): 775-802.

[31] Godefroid P. Compositional dynamic test generation [C]//Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, USA: ACM Press, 2007: 257-269.

[32] Godefroid P, Levin M, Molnar D. Automated white-box fuzz testing [C]//Proceedings of the 10th International Conference on Network and Distributed Systems Security. San Diego, USA: Schloss Dagstuhl, 2008: 201-213.

[33] Hatcliff J, Dwyer M. Using the Bandera tool set to model-check properties of concurrent Java software [C]//Proceedings of the 12th International Conference on Concurrency Theory. London, UK: Springer-Verlag, 2001: 351-365.

[34] CHEN Hao, Wagner D. MOPS: An infrastructure for examining security properties of software [C]// Proceedings of the 9th ACM Conference on Computer and Communications Security. New York, USA: ACM Press, 2002: 179-191.

[35] Beyer D, Henzinger T, Jhala R, et al. The software model checker blast [J]. *International Journal on Software Tools for Technology Transfer*, 2007, **9**(56): 505-525.

[36] Microsoft Research. SLAM [Z/OL]. (2010-08-14), http://research.microsoft.com/slam/.

[37] Havelund K, Pressburger T. Model checking Java programs using Java path finder [J]. *International Journal on Software Tools for Technology Transfer*, 2000, **2**(4): 366-381.

[38] Flanagan C, Leino K, Lillibridge M, et al. Extended static checking for Java [C]//Proceedings of the 24th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2002: 282-294.

[39] XIE Yichen, Aiken A. Saturn: A scalable framework for error detection using Boolean satisfiability [J]. *ACM Transactions on Programming Languages and Systems*, 2007, **29**(3): 164-225.

[40] 陈石坤，李舟军，黄永刚，等. 一种基于 SAT 的 C 程序缓冲区溢出漏洞检测技术 [J]. 清华大学学报：自然科学版，2009，**49**(S2)：2165-2175.
CHEN Shikun, LI Zhoujun, HUANG Yonggang et al. SAT-based technique to detect buffer overflows in c source codes [J]. *Journal of Tsinghua University: Science and Technology*, 2009, **49**(S2): 2169-2175. (in Chinese)

[41] Flanagan C, Leino K. Houdini, An annotation assistant for ESC/Java [C]//Proceedings of the 4th International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity. London, UK: Springer-Verlag, 2001: 500-517.

[42] **忽朝俭, 李舟军, 郭涛, 等. 写污点值到污点地址漏洞模式检测** [J]. **计算机研究与发展**, 2011, **48**(08): 1455-1463.
HU Chaojian, LI Zhoujun, GUO Tao et al. Detecting the vulnerability pattern of writing tainted value to tainted Address [J]. *Journal of Computer Research and Development*, 2011, **48**(08): 1455-1463. (in Chinese)

[43] Jovanovic N, Kruegel C, Kirda E. Pixy: A static analysis tool for detecting Web application vulnerabilities [C]// Proceedings of the 27th IEEE Symposium on Security and Privacy (S&P'06). Piscataway, USA: IEEE Press, 2006: 221-236.

[44] Jovanovic N, Kruegel C, Kirda E. Precise alias analysis for static detection of Web application vulnerabilities [C]// Proceedings of the 1st ACM SIGPLAN Workshop on Programming Languages and Analysis for Security. New York, USA: ACM Press, 2006: 508-520.

[45] Fortify Corporation. Fortify [Z/OL]. (2011-09-11), http://www.fortify.net/.

[46] **严俊, 郭涛, 阮辉, 等.** JUTA: **一个** Java **自动化单元测试工具** [J]. **计算机研究与发展**, 2010, **47**(10): 1840-1848.
YAN Jun, GUO Tao, LUAN Hui et al. JUTA: An automated unit testing framework for Java [J]. *Journal of Computer Research and Development*, 2010, **47**(10): 1840-1848. (in Chinese)

[47] **程绍银, 蒋凡, 王嘉捷, 等. 一种自动生成软件缺陷输入的方法** [J]. **中国科学技术大学学报**, 2010, **40**(2): 191-196.
CHEN Shaoyin, JIANG Fan, WANG Jiangjie et al. A Method for automatically generating inputs of software bug [J]. *Journal of University of Science and Technology of China*, 2010, **40**(2): 191-196. (in Chinese)

[48] Sutton M, Greene A, Amini P. Fuzzing: Brute Force Vulnerability Discovery [M]. Boston, USA: Addison-Wesley Professional, 2007.

[49] INFOSEC Corporation. SPIKE [Z/OL]. (2012-09-14), http:// resources.infosecinstitute.com/intro-to-fuzzing/.

[50] IOACTIVE. Peach [Z/OL]. http://peachfuzzer.com/.

[51] WANG Tielei, WEI Tao, GU Guofei, et al. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection [C]// Proceedings of the 31st IEEE Symposium on Security & Privacy. Piscataway, USA: IEEE Press, 2010: 107-121.

[52] Song D, Brumley D, Yin M, et al. BitBlaze: A new approach to computer security via binary analysis [C]// Proceedings of the 4th International Conference on Information Systems Security. New York, USA: ACM Press, 2008: 147-162.

[53] Monga M, Paleari R, Passerini E. A hybrid analysis framework for detecting web application vulnerabilities [C]// Proceedings of the 6th ICSE Workshop on Software Engineering for Secure Systems. Piscataway, USA: IEEE Press, 2009: 202-219.

[54] Silvius R, Lawrence R, Jay H. Hybrid analysis: Static & dynamic memory reference analysis [C]// Proceedings of the 16th International Conference on Supercomputing. Piscataway, USA: IEEE Press, 2002: 311-325.

[55] Tlili S, YANG Zhenrong, LING Haizhou, et al. A hybrid approach for safe memory management in C [C]//Proceedings of the 12th International Conference on Algebraic Methodology and Software Technology. New York, USA: ACM Press, 2008: 315-328.

[56] DONG Guowei, WU Shizhong, WANG Guisi, et al. Security assurance with metamorphic testing and genetic algorithm [C]// Proceedings of the 11th IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. Piscataway, USA: IEEE Press, 2010: 397-401.

[57] Zhang R, Huang S, Qi Z, et al. Combining static and dynamic analysis to discover software vulnerabilities [C]// Proceedings of the 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. Piscataway, USA: IEEE Press, 2011: 101-122.

[58] Ruben E, Roger K, Michael Z. AWE: Improving software analysis through modular integration of static and dynamic analyses [C]// Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software tools and Engineering. New York, USA: ACM Press, 2007: 299-310.

[59] Pranith D, Anchal D, Rajeev K. Hybrid analysis of executables to detect security vulnerabilities: security vulnerabilities [C]// Proceedings of the 2nd India Software Engineering Conference. Piscataway, USA: IEEE Press, 2009: 379-392.

[60] Oleh Yuschuk. OllyDbg [Z/OL]. (2012-01-01), http://www.ollydbg.de/.

[61] Hex-Rays SA. IDA [Z/OL]. (2011-03-28), http://www.hex-rays.com/products/ida/index.shtml.

[62] Asaf Yaffe. PIN [Z/OL]. (2011-07-11), http://www.pintool.org/.

[63] Seward J. Valgrind [Z/OL]. (2011-04-23), http://valgrind.org/.

[64] LIAO Chunhua, Quinlan D, Vuduc R, et al. Effective source-to-source outlining to support whole program empirical optimization [C]// Proceedings of the 22nd international conference on Languages and Compilers for Parallel Computing. New York, USA: ACM Press, 2010: 197-211.

[65] Schordan M, Quinlan D. A source-to-source architecture for user-defined optimizations [J]. *Lecture Notes in Computer Science*, 2003, **2789**(10): 214-223.

[66] **王金锭, 王嘉捷, 程绍银, 等. 基于统一中间表示的软件漏洞挖掘系统** [J]. **清华大学学报: 自然科学版**, 2010, **50**(S1): 1502-1507.
WANG Jinding, WANG Jiajie, CHENG Shaoyin et al. Software vulnerability detection system based on uniform intermediate representation [J]. *Journal of Tsinghua University: Science and Technology*, 2010, **50**(S1): 1502-1507. (in Chinese)

[67] Walker D. A type system for expressive security policies [C]// Conference Record of POPL'00: 27th ACM SIGPLAN-SINACT. Symposium on Principles of Programming Languages. New York, USA: ACM Press, 2000: 254-267.

[68] Evans D, Larochelle D. Improving security using extensible lightweight static analysis [J]. *IEEE Software*, 2002, **19**(1): 42-51.

[69] Ashcraft K, Engler D. Using programmer-written compiler extensions to catch security holes [C]// Proceedings of the 23rd IEEE Symposium on Security and Privacy. Piscataway, USA: IEEE Press, 2002: 143-159.

[70] Conway C. Incremental algorithms for inter-procedural analysis of safety properties [J]. *LNCS*, 2005, **3576**(2): 11-22.

[71] Silva V, Kroening D, Weissenbacher G. A survey of automated techniques for formal software verification [J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008, **27**(7): 1165-1178.

[72] Cadar C, Dunbar D, Engler D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs [C]// Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. Berkeley, USA: USENIX Association, 2008: 209-224.

[73] Boonstoppel P, Cadar C, Engler D. RWset: Attacking path explosion in constraint-based test generation [J]. *Tools and Algorithms for the Construction and Analysis of Systems*, 2008, **4963**(2): 351-366.

[74] Daniel J, Richard W, Ghassan M. Techniques for specifying bug patterns [C]// Proceedings of the 2007 ACM Workshop on Parallel and Distributed Systems: Testing and Debugging. New York, USA: ACM Press, 2007: 97-112.

[75] Udrea O, Lumezanu C, Foster J. Rule-based static analysis of network protocol implementations [J]. *Information and Computation*, 2008, **206**(2): 130-157.

[76] Patrice G. Random testing for security: Blackbox vs. whitebox fuzzing [C]//Proceedings of the 2nd International Workshop on Random Testing: Co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering. New York, USA: ACM Press, 2007: 329-341.

[77] Bucur S. Parallel symbolic execution for automated real-world software testing [C]// Proceedings of the 6th Conference on Computer Systems. New York, USA: ACM Press, 2011: 107-122.

[78] ZHOU Wu, ZHOU Yajin, JIANG Xuxian, et al. DroidMOSS: Detecting repackaged smartphone applications in third-party Android marketplaces [C]// Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy. New York, USA: ACM Press, 2012: 215-224.

[79] Grace M, ZHOU Yajin, WANG Zhi, et al. Systematic detection of capability leaks in stock Android smartphones [C]// Proceedings of the 19th Network and Distributed System Security Symposium. Piscataway, USA: IEEE Press, 2012: 316-330.

[80] Enck W. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones [C]// Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation. New York, USA: ACM Press, 2010: 313-328.

[81] Godefroid P, Molnar D. Fuzzing in the Cloud [M]. Seattle, USA: Microsoft Press, 2010.

[82] Zalewski J. Can we measure security and how? [C]// Proceedings of the 7th Annual Workshop on Cyber Security and Information. New York, USA: ACM Press, 2011: 46-52.