doi :10.3969/j.issn.1671-1122.2017.08.001

二进制程序漏洞挖掘关键技术研究综述

王夏菁[1]，胡昌振[1]，马锐[1]，高欣竺[2]

（1. 北京理工大学软件学院软件安全工程技术北京市重点实验室，北京 100081；2. 中国黄金集团公司，北京 100011）

摘　要：漏洞在当前的网络空间中已被各方所关注。虽然源代码漏洞分析取得显著进展且具有语义丰富的特点，但实际应用中大量软件均以二进制代码形式存在，因此，针对二进制代码的漏洞挖掘技术研究具有很强的实用价值。文章简要介绍了目前较为典型的二进制漏洞分析框架，并根据现有研究工作，提出未来对二进制程序漏洞挖掘技术研究的整体思路，随后对其中的一些关键点、关键技术分别进行了调研。文章首先对中间语言的研究背景和意义进行了简要介绍；其次针对污点分析、符号执行以及模糊测试三项关键技术，分别介绍了三者的基本原理和分类标准、处理流程、研究现状以及存在的问题；最后进行了简单的总结。文章对二进制程序的漏洞挖掘技术进行了简要的研究，有助于开展后续研究工作。

关键词：漏洞挖掘；二进制程序；污点分析；符号执行；模糊测试

中图分类号：TP309.1　　**文献标识码：**A　　**文章编号：**1671-1122（2017）08-0001-13

中文引用格式：王夏菁，胡昌振，马锐，等 . 二进制程序漏洞挖掘关键技术研究综述 [J]. 信息网络安全，2017（8）：1-13.

英文引用格式：WANG Xiajing, HU Changzhen, MA Rui, et al. A Survey of the Key Technology of Binary Program Vulnerability Discovery[J]. Netinfo Security, 2017(8):1-13.

A Survey of the Key Technology of Binary Program Vulnerability Discovery

WANG Xiajing[1], HU Changzhen[1], MA Rui[1], GAO Xinzhu[2]

(1.*Beijing Key Laboratory of Software Security Engineering Technology*, *School of Software*, *Beijing Institute of Technology*, *Beijing* 100081, *China*; 2.*China National Gold Group Corporation*, *Beijing* 100011, *China*)

Abstract: In the current cyberspace, vulnerability has been attracted the widespread attention. Although source-code-oriented vulnerability analysis has made significant progress and has the characteristics of rich semantic, but many commercial software exists in the form of binary code in practical application. Therefore, binary-executable-oriented vulnerability discovery is more meaningful and useful. This paper first briefly introduces the typical binary vulnerability analysis framework. Based on the existing research work, this paper puts forward the whole idea of the research on the vulnerability discovery technology of binary program in the future, and then makes some research on some key points and key technologies respectively. Firstly, this paper briefly introduces the research on the key technologies of binary-executable-oriented background and significance of the intermediate language. Secondly, according to the three key technologies of taint analysis, symbolic execution and fuzzing, this paper introduces the basic principles and classification standards, processing flow, research situation and existing problems, respectively, and finally gives a simple summary. In this paper, a brief study of the binary program vulnerability discovery technology is given, which is helpful to carry out the follow-up research work.

Key words: vulnerability discovery; binary program; taint analysis; symbolic execution; fuzzing

收稿日期：2017-6-22

基金项目：国家重点研发计划 [2016QY07X1404]

作者简介：王夏菁（1994—），女，山西，博士研究生，主要研究方向为信息安全；胡昌振（1967—），男，湖北，教授，博士，主要研究方向为信息安全；马锐（1972—），女，河南，副教授，博士，主要研究方向为信息安全；高欣竺（1982—），女，山东，助理工程师，硕士研究生，主要研究方向为网络安全。

通信作者：马锐 mary@bit.edu.cn

0 引言

随着信息科技革命热潮在全球的快速兴起，信息技术已经影响到个人、社会和国家的方方面面。与此同时，各种安全事件也层出不穷，成为全世界面临的主要问题之一[1]。例如，中国最大的程序员社区网站 CSDN，曾在 2011 年被暴出有超过 600 万用户的注册信息，尤其是用户口令遭到泄露。这种类似的用户数据信息泄露事件给公民的个人信息安全带来了威胁。攻击者借助某个漏洞肆意地传播恶意病毒和文件，软件漏洞所导致的蠕虫、网马（Driven-by Downloads）、僵尸网络（Botnet）等各种网络攻击事件给世界各国的社会和经济带来严重危害。从本质上而言，这些攻击事件大多是通过对软件漏洞的利用而引发的，正是因为这些漏洞的存在，才使得攻击者得以获取终端的控制权来运行恶意代码或植入后门。因此，漏洞在信息安全中处于核心地位，并且在当前的网络空间中被各方所关注。

软件漏洞产生的根源是现代计算机在实现图灵机模型时，没有在内存中将程序代码和数据严格地区分开，内存中的数据也能被程序当作代码执行[2]。正是基于这个原因，攻击者才能通过一些手段来构造恶意程序，并成功植入目标程序进行攻击，进而获取目标程序的控制权。面对层出不穷的攻击手段，学术界和工业界都在软件漏洞分析领域展开了大量的相关研究工作，提出了许多解决安全问题的方法和工具。

在程序语言设计以及程序分析等技术迅速发展的推动下，面向源代码的软件漏洞挖掘技术已经取得了显著进展[3]。然而面向源代码的软件漏洞挖掘技术也存在一些不足。首先，大多数软件并未提供程序源代码；其次，高级语言以及核心操作系统最终也都会被编译成二进制代码，并且高级语言在进行编译、链接的过程中也可能会引入漏洞，使得由此而引入的漏洞难以检测。而面向二进制程序的漏洞挖掘研究因其具有语言无关性且不需要程序源代码，同时也不需要进行编译和链接，可以直接被执行，因此直接对二进制代码的漏洞研究更加实用。

但是，面向二进制程序的漏洞分析也同样面临一些挑战。最大的问题就是底层指令较为复杂，且程序缺少包含语义和语法的类型信息，导致难以理解，因此研究人员必须具备大量的计算机底层知识。基于此问题，业界侧重于将二进制程序翻译成中间语言[4]，通过中间语言获取需要的语义和类型信息，之后借助这些信息进行漏洞分析的研究工作。中间语言具备指令精简以及语义丰富的优势，因此弥补了直接对底层指令进行分析的不足。

现有的二进制漏洞分析技术有多种，其分类方式也有多种[1]。从操作的自动化角度可分为手工或自动 / 半自动化分析；从软件运行角度可分为动态分析、静态分析和动静结合分析；从软件代码的开放性角度可分为黑盒测试、白盒测试和灰盒测试；从研究对象形态的角度可分为基于中间语言和基于底层指令集的漏洞分析技术。由于是从不同的角度进行划分的，因此无论按照哪种标准进行分类，各种分类之间都有交叉和重叠，其具体实现技术也可以同时属于不同分类标准下的不同类别。典型的具体技术包括模糊测试、符号执行和污点分析，这 3 种技术几乎涵盖了这些分类标准。污点分析和符号执行两者均既属于从软件运行角度下划分的动态分析和静态分析，同时也可以是从代码开放性角度下划分的白盒测试。模糊测试既属于动态分析，也属于黑盒测试。此外，以上 3 种技术均可以是基于中间语言和基于底层指令的漏洞分析技术。

因此，虽然目前的二进制漏洞分析技术有多种，但是一些典型的具体技术几乎可以代表整个二进制漏洞分析技术。污点分析从软件漏洞利用的根本出发，标记不可信的输入作为污点数据，跟踪污点数据在程序执行过程中的使用情况，如果程序中的关键操作非法使用了污点数据，则判断程序中存在漏洞。由于其抓住了漏洞利用的实质，因此该方法在原理上是有效的[2]。与此同时，随着硬件计算能力的提高，符号执行技术由于路径覆盖率很高，得到越来越广泛的应用。相比于动态污点分析的关注点为数据流，符号执行则是对控制流的分析。因此，这两项技术在应用于漏洞分析方面具有很好的互补性。此外，模糊测试作为软件漏洞分析的代表性技术，因其具有不依赖于程序源代码且系统消耗低的优点，所以在软件漏洞分析领域占据重要地位。

由于现阶段大多数的二进制漏洞分析工具没有集成在一个平台上，之前研究所得的成果在后期往往难以相互协作和重用，后续研究者也无法在此基础上进行扩充，只能重新实现，造成了很多资源浪费。为了解决这个问题，一

个较好的思路是结合现有的分析技术，并将它们集成于一个平台，形成一个综合的二进制漏洞分析框架。因此在未来研究工作中，将会根据现有的分析框架，构造一个更为强大的二进制分析框架。

1 二进制程序漏洞分析框架

针对软件中存在的漏洞，研究人员提出了很多分析技术并且已经广泛地应用于漏洞分析领域。但是现有的研究工作由于大多是使用某种分析工具对某种特定类型的程序进行漏洞分析，或者是对这些分析技术的优化，因此会面临以下问题。首先，很多二进制分析技术的研究工作难以重用，后续的研究者不能在前人研究的基础上进行扩展和补充，只能根据前人提出的方法重新实现，这意味着很多研究工作被浪费；其次，使用单一的分析技术不能实现优势互补。由于每种技术都有其优点和不足，有些分析技术面临的一些关键问题至今无法解决，制约了这些分析技术的发展，因此通过结合两种或者多种分析技术进行优势互补将是未来的研究方向。

基于目前研究工作中存在的问题，较好的解决方法是构建一个统一的二进制漏洞分析框架，并集成一些主要的分析技术，以便后续研究者可以在分析框架的基础上优化或集成下一代二进制分析技术，实现技术上的重用。

目前已经存在一些二进制漏洞分析框架。例如，SONG[5] 等人提出的 BitBlaze。BitBlaze 是一个统一的二进制分析平台，结合了动态分析和静态分析，并且具有可扩展性。BitBlaze 主要包含 3 个组件，分别是 Vine、TEMU 和 Rudder。Vine 是静态分析组件，其将底层指令翻译成简单且规范的中间语言，并且在中间语言的基础上为一些常见的静态分析提供了实用工具，如绘制程序依赖关系图、数据流图及程序控制流图等；TEMU 是动态分析组件，其提供了整个系统的动态分析，并且实现了语义提取和用户定义动态污点分析；Rudder 是结合动静态分析的具体执行和符号执行组件，其使用 Vine 和 TMEU 提供的功能在二进制层面上实现了混合具体执行和符号执行，并且提供了路径选择和约束求解的功能。BitBlaze 中组件的主要处理流程如图 1 所示。

SHOSHITAISHVILI[6] 等人提出了一个多架构的二进制

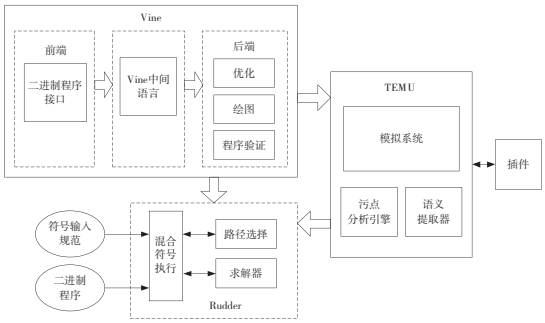

图 1 BitBlaze 处理流程图

分析框架 angr，集成了很多二进制分析技术，具备对二进制程序的动态符号执行能力和静态分析能力。angr 是由 shellfish 团队在 CTF 比赛中开发的二进制自动化分析工具，起初用于寻找程序中的后门，现在可以应用于漏洞分析领域。由于 angr 集成了一些现有的分析技术，同时使用不同的模块实现不同的功能，因此，可以很容易地在平台上对已有的分析技术进行比较，并且能利用不同分析技术的优势。其简要的处理过程是：首先，将二进制程序加载到 angr 分析平台中；然后，将二进制代码转换成中间语言；最后，进一步分析程序，其中包括静态分析或对程序的符号执行进行的探索。angr 处理流程图如图 2 所示。

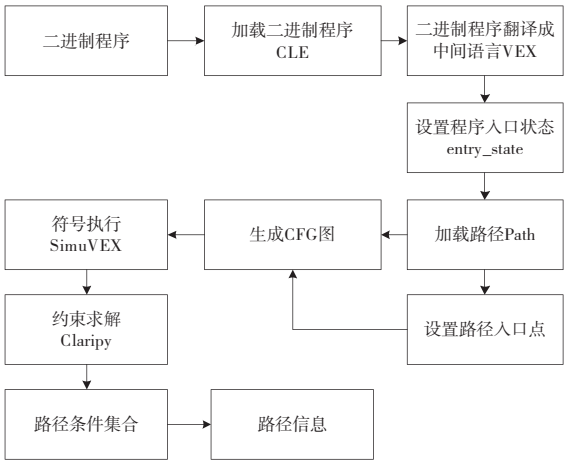

图 2 angr 处理流程图

angr 主要包括以下几个模块：中间表示模块（IR），将二进制代码翻译成中间语言，其中间语言 VEX 可以在不同架构上分析二进制程序；二进制程序加载模块（CLE），将一个二进制程序加载到分析平台中；程序状态表示模块（SimuVEX），表示程序的状态，并且 SimuVEX 中的 SimState 实现了一组状态插件的集合，如寄存器、抽象内存及符号内存等，这些插件的状态可以由用户指定；数据模型模块（Claripy），为存储在 SimState 的寄存器或存储

器中的值提供抽象表示；完整程序分析模块，将所有的模块组合起来使得 angr 可以进行复杂且完整的程序分析。

angr 是一个开源的工具，其兼容性较好，同时支持跨平台、跨架构，并且提供了强大的符号执行引擎，具有很强的分析能力。但是，由于 angr 设计的初衷是为了寻找程序中的后门，因此如果将其应用在漏洞分析领域，目前还无法进行完整的漏洞分析。首先，在符号执行部分，angr 仅提供了程序的路径信息，对于后续的分析需要人工辅助进行；其次，angr 目前只是针对符号执行部分较为强大，还未集成其他较为典型的分析技术。因此，实现后续分析过程的自动化以及集成其他分析技术作为辅助需要进行进一步研究。

在未来的研究工作中，将会结合目前的热点技术，构建一个综合的二进制漏洞分析平台，从而形成一个规模化的二进制漏洞分析框架。具体构建思路如下：1）增加对二进制程序的建模以及在此基础上进行形式化的描述，从而建立安全策略库。2）对中间语言部分进行优化，针对二进制程序转换成中间语言过程中所面临的信息丢失以及转换效率较低的问题，将会进一步补充和完善，希望能够获取更加准确的信息流（具体包括数据流和控制流）。3）集成符号执行并对该技术进行进一步优化，实现后续分析的自动化；同时集成污点分析技术，用来辅助符号执行减少路径数量，从而减缓路径爆炸的问题。此外，污点分析也可以结合上述建立的安全策略库进行漏洞分析。4）集成模糊测试技术，并根据约束求解器求解出每条路径的约束条件继而有针对性地生成测试用例，然后结合模糊测试判断漏洞。二进制漏洞分析框架的总体流程如图 3 所示。

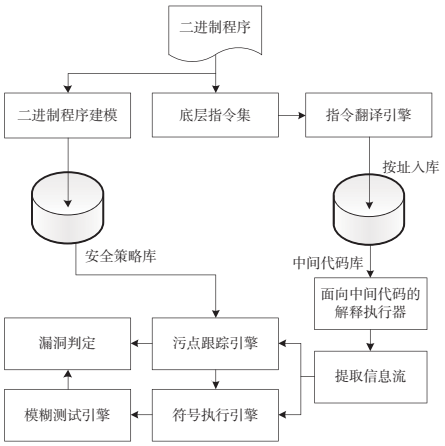

图 3 二进制漏洞分析框架的总体流程图

二进制漏洞分析技术种类众多，应用场景各不相同，在此选择 3 个主流且有代表性的分析技术，即污点分析、符号执行及模糊测试技术，将三者结合，共同集成到一个综合的二进制漏洞分析平台，进而进行漏洞分析。本文后续会单独介绍这 3 种分析技术的研究现状。

2 中间语言研究背景及意义

与面向源代码的漏洞挖掘技术相比，面向二进制代码的漏洞挖掘技术更具研究价值，主要因为它具有不依赖于源码的特性，实用性更广泛。但由于面向二进制程序的漏洞分析技术特有的复杂性，直接分析二进制程序比较困难。主要存在以下两个原因：1）底层指令集数量较多，如 x86 的指令集有数百个，而且较为复杂；2）一些简单的问题，如从数据中分离代码，是不可判定的，也就是说二进制代码缺乏相应的语义和类型信息。

基于上述困难，业界研究者更希望能以一种更容易的方式来表示二进制程序，因此提出了使用中间语言 [4] 来替代二进制代码，即把二进制代码转化成具有语义信息的中间代码，以便于研究人员的后续分析。这就要求使用的中间语言不仅要比高级语言层次低以便从二进制代码转换成中间语言不会很复杂，也需要比二进制程序高级以便能够提供大量的语义信息。中间语言本身是底层语言的进一步规范，其主要是服务于之后的漏洞分析。转化中间语言是进行二进制程序漏洞分析的最重要的一步，需要通过中间语言所具有的语义信息进行分析以便能够获取程序的 CFG 图，从而获取程序的信息流，其中包括控制流和数据流。因此，得到由二进制代码翻译而来的中间语言是进行后续工作的基础。

针对如何转化中间语言，目前存在不少方法和工具。陈凯明 [7] 等人提出采用符号执行技术进行转化的方法，首先需要定义符号执行的环境和语义，然后在符号环境下执行汇编语言，转换成对应的中间语言。由于其需要像符号执行一样模拟程序执行，因此会增加开销。姜玲燕 [4] 等人提出了一种称为 VINST 的中间表示方法，其遵循使执行频繁的部分保持高效，使其他部分保持正确的基本原则，因此只包含了各种指令集中最常用的二十余条指令，剩下的相对少用的简单指令用多种中间指令分别对应其

功能进行模拟，复杂指令用 CALL 中间指令，让高级语言的函数来模拟，这种方法形式比较简单，但是效率很低。CIFUENTES[8] 等人开发了二进制翻译系统 UQBT，其描述系统的指令信息是通过语义规范语言 SSL 来实现的，同时通过生成指令字典来完成从汇编语言到中间语言的转换。由于其中间语言的生成完全依赖于 SSL 语言对指令系统的描述，因此如果没有对 SSL 进行准确的描述，根据其转换成的中间语言会存在语义缺失或者不准确的问题。随后的反编译器 Boomerang[9] 是在 UQBT 的基础上开发的，因此与 UQBT 的原理类似。马金鑫 [10] 等人针对类型重构引入了一种具有静态单赋值形式的中间语言，其中，对每个变量的定义只进行一次，从而把庞大复杂的 x86 汇编指令归约为简单的等式形式，并在此基础上提出了一种构造寄存器抽象语法树的方法，解决了基址指针别名问题。但是由于在翻译过程中忽略了多种情况，因此这种方法翻译出的中间语言存在信息丢失的问题。Valgrind[11] 平台也采用了一种中间表示，即 VEX 中间语言，与架构无关。VEX 采用 RISC 指令集，其平台中有针对内存和寄存器的读写操作。内存变量中的 LOAD 和 STORE 分别是读取和写入操作；寄存器变量中的 GET 操作和 PUT 操作，两者分别负责将寄存器变量中的值读取到临时变量中，或将临时变量中的值写入到寄存器变量；IMark 语句是对某一条具体指令地址的标记。

但是，目前基于中间语言的漏洞分析技术也面临一些问题。首先，翻译成中间语言的过程中会出现丢失部分信息的情况，如 Valgrind 使用的 VEX 中没有保存 EFLAGS 寄存器 [11]；其次，中间语言会将一条指令翻译成若干条指令，可能会降低效率，如 Valgrind 使用的 VEX、BitBlaze 使用的 VINEIL 等中间语言。

由于中间语言是后续工作的前提，因此如何提高二进制翻译程序的翻译效率和准确率是每个翻译程序急需解决的问题。针对中间语言部分，未来的研究工作主要包括以下两个方面：首先，针对中间代码会带来很多冗余信息的问题，在设计的过程中需要以简明扼要为首要原则，尽量减少重复的信息，每条指令以实现单一功能为主要目标；其次，在减少冗余信息的前提下，需要尽量保证所翻译的中间语言具有完整的语义信息。因此针对中间语言目前存在的信息丢失的问题，需要进一步进行优化。通过将中间语言与所对应的二进制程序反汇编得到的汇编语言进行比较，调研中间语言是否缺少一些重要的语义信息，同时需要对中间语言缺少的语义信息进行补充和完善，以便为后续漏洞分析提供尽可能完整的数据流和控制流。

3 动态污点分析技术

3.1 动态污点分析技术原理和分类

污点分析技术是一种跟踪并分析污点信息在程序中流动的技术，最早由 DENNING[12] 于 1976 年提出。污点分析技术的主要原理是将来自于网络、文件等非信任渠道的数据标记为“被污染的”，则作用在这些数据上的一系列算术和逻辑操作而新生成的数据也会继承源数据的“被污染的”属性 [1]。通过对这些属性进行分析，进而得出程序的某些特性。污点分析技术从是否执行程序的角度一般可分为动态污点分析和静态污点分析两种。

静态污点分析技术的优点是能快速定位污点在程序中可能出现的所有情况，但是因为静态分析无法获得程序运行时的一些信息，存在精度较低的问题，因此往往需要人工对分析结果进行复查确认 [13]。通常情况下，二进制静态污点分析技术是指利用反汇编工具，如 IDA，在对二进制代码进行反汇编的基础上使用静态污点分析的技术。

动态污点分析技术是近年来逐渐流行的另一种污点分析技术，也称为动态信息流分析。该技术是在程序运行时标记和追踪程序中的特定的数据，依据这些特定的数据是否会影响某些事先约定好的关键操作来判定程序是否存在漏洞。在应用程序的安全性领域中，已经成功地使用动态污点分析来防止多种类型的攻击，如缓冲区溢出、格式字符串攻击、SQL 命令注入和跨站点脚本攻击（XSS）等。最近，研究人员已经开始研究将基于污点分析的方法应用在安全以外的领域，如理解程序、软件测试和调试等。

动态污点分析技术按分析粒度可以划分为细粒度动态污点分析技术和粗粒度动态污点分析技术 [14]。粗粒度污点分析一般可用于检测异常行为，其优势在于分析速度较快、占用存储空间小，但是往往会存在分析精度低的问题；细粒度污点分析主要用于检测程序的脆弱性攻击点，其优势在于分析精度高、可以解决数据流回溯等问题，但是会存在测试代价大及占用空间过大的难题。史大伟 [14] 等人

将两者进行结合，优势互补，提出了一种粗细粒度结合的动态污点分析方法。

3.2 动态污点分析基本过程

动态污点分析的主要过程由 3 个阶段组成，如图 4 所示。

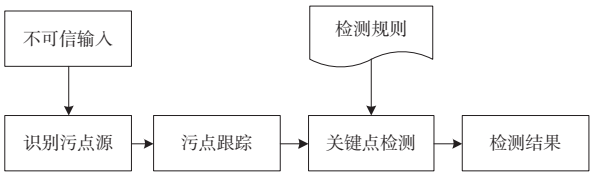

图 4 动态污点分析的基本过程

1）识别污点源

识别污点源是污点分析的前提。目前的污点源标识方法主要有：将程序的全部外部输入都标记为污点数据；根据人工分析进行手工标记污点源；使用统计或者机器学习技术自动识别和标记污点源。

2）污点跟踪

根据特定的传播规则对污点数据进行跟踪，查看污点数据的传播路径。污点传播规则是污点分析过程中最重要的部分。对于二进制代码的污点传播分析有两种方法，一种是直接分析底层指令集，另一种是分析二进制代码翻译后的中间语言[15]。由于直接对二进制代码分析缺乏语义信息，分析难度较大，因此将采用基于中间语言的污点分析技术，根据中间语言获取的信息流进行污点分析。

3）关键点检测

根据合理的检测规则对关键点进行检测，查看程序的关键点是否使用了带有污点标记的数据。

3.3 研究现状

近年来，国内外很多学者都围绕污点分析方法进行了较为深入的研究，YIN[16] 等人提出了一个基于 QEMU 虚拟机的扩展平台 TEMU，可以解决动态分析的一些困难，便于在其上进行程序分析。TEMU 使用全系统的视角，可分析内核中的活动与多进程间的交互，且以细粒度的方式进行深度分析。KANG[17] 等人提出了一种动态的污点传播方案 DTA++，包括两个阶段：首先，通过诊断产生不完全污点的分支生成规则，并确定离线分析所需额外的传播；其次，在以后的动态污点分析中应用那些规则。其基本原理是：查找不完全污点的控制流的路径，同时使用符号执行和基于路径谓词的方法求解该路径的约束条件，在符号执行中，程序的执行轨迹可以表示为一系列分支条件判断的路径约束集合。诸葛建伟[2] 等人提出了基于类型感知的动态污点分析技术和面向类型变量的符号执行技术，有效解决了目前工作在二进制层次上的污点分析技术无法为软件漏洞分析提供高层语义的问题。该技术将标记 Type Source 点的污点变量，添加类型信息作为其污点属性，构成污点源。在污点传播的过程中，再利用指令、公开库函数的语义，结合 Type Sink 点揭示的类型信息作为补充，将变量的类型信息进行传递，做变量级污点传播，且在污点传播的同时进行面向类型变量的符号执行，获取程序执行路径条件，形成安全漏洞特征库。马金鑫[18] 等人提出了基于执行踪迹离线索引的污点分析方法，使用动态插桩工具跟踪程序执行，把执行的状态信息记入踪迹（trace）文件中；然后离线地分析踪迹文件，并建立索引文件，在污点传播中只关注与污点数据相关的操作，与污点数据无关的指令直接跳过，从而节省了分析时间。

3.4 存在的问题

在目前的动态污点分析研究中，主要存在以下几方面问题。

3.4.1 隐式流问题

隐式流问题就是控制流依赖关系下的污染问题。在污点传播分析过程中，根据所关注的程序依赖关系的不同，可以分为显式流分析和隐式流分析。显式流分析是分析污点标记随着程序中变量之间的数据依赖关系传播；隐式流分析是分析污点标记随程序变量之间的控制依赖关系传播。隐式流污点传播是污点分析技术中一个比较重要的问题[19]，如果没有对污点数据进行适当的处理，会造成污点分析的结果不精确。目前主要存在两种情况，即污点数据的欠污染（under-taint）和过污染（over-taint）。欠污染问题是由于没有对隐式流污点传播进行适当的处理，导致本应被标记的变量没有被标记。过污染问题是因为所标记的污点数量过多而导致污点变量大量扩散。目前针对隐式流问题的研究重点是尽量减少欠污染和过污染的情况。

动态隐式流所关注的第一个问题是如何确定污点控制条件下需要进行标记的语句的范围。由于动态执行轨迹并不能反映出被执行程序之间的控制依赖关系，因此目前采

用的方法是使用离线静态分析辅助判断动态污点传播中的隐式流标记范围。动态分析所面临的第二个问题是由于部分污点信息泄露导致潜在安全问题的漏报。由于在动态污点分析时，会存在没有覆盖的路径，即未被执行的路径，因此会出现污点信息通过这些未被执行的路径进行传播并泄露的情况。目前所采取的解决方法也是在动态分析的基础上增加离线的静态分析。第三个问题是如何选择合适的污点标记分支进行污点传播。如果简单地将所有包含污点标记的分支都进行传播，将会导致过污染的情况，因此需要对污点标记分支进行筛选标记，减少过污染。KANG[17] 等人提出的 DTA++ 工具使用基于离线执行踪迹的符号执行方法来寻找进行污点传播的分支。

3.4.2 污点消除问题

在污点分析中，如果污点标记的数量一直增加，并在执行过程中连续传播，有些应当被清除的污点没有进行消除，就会导致误报，从而影响分析结果的准确度 [20]。正确地使用污点消除可以降低系统中污点标记的数量，提高污点分析的效率，并且避免由于污点扩散导致的分析结果不精确的问题。因此，对于一些特殊情况，如存在对敏感数据进行加密处理以及存在常数函数等情况，即可考虑消除污点标记。

3.4.3 分析代价较大

部分污点分析工具需要使用插桩或动态代码重写技术，会给分析系统带来巨大的开销 [19]。例如，TaintCheck[21] 利用插桩工具 Valgrind 对其中间表示 Ucode 插桩并实现针对 x86 程序的动态污点分析；Dytan[22] 使用插桩程序 Pin 实现针对 x86 程序的动态污点分析。为了控制分析代价，现存的研究工作采用的思路是有选择地对系统中的指令进行污点传播分析。例如，QIN[23] 等人提出的快速路径优化技术，通过提前判断某个模块的输入输出是否存在威胁，以此判断是否需要进行污点传播，如果没有威胁就不进行污点传播，从而降低开销。彭建山 [24] 等人通过建立可疑污点集合以缩小待分析污点范围，利用污点回溯技术（一种反向的污点传播分析）追踪污点来源，从而减小开销。林伟 [25] 等人进一步解决了离线污点分析中的针对轨迹记录文件进行污点传播分析的时间开销大的问题，提出了基于语义规则的高效的污点传播方法，从而减小开销，提高了效率。但是这些方法或多或少会对分析精度有所影响，因此还需要进一步研究如何在不降低分析精度的前提下降低分析代价。

3.4.4 漏报率较高

动态污点分析技术的漏报率较高、精度较低是较难解决的问题，因为它是由程序动态运行的性质决定的。任何程序单次运行时，其代码的执行过程都可能发生变化，这与向程序输入的测试用例不同有关。因此，动态污点分析技术单次的分析结果可能仅仅覆盖了程序的部分代码，从而使得其无法挖掘部分未覆盖代码，造成较高的漏报率。要解决这个问题，需要对整个程序代码和功能进行分析，通过不断地构造合适的测试用例来尽量使动态分析的代码覆盖率趋近于 100%。朱正欣 [26] 等人为了解决污点分析漏报率较高的问题，提出了动态符号化污点分析方法，该方法使用符号化的思想将污点信息及风险规则进行符号化，同时检测污点数据在传播过程中是否存在违反某些风险规则的行为，进而检测出程序的不安全行为。崔化良 [27] 等人提出了离线污点分析方法，将污点分析系统拆分为两个模块（动态记录模块和静态重放模块），通过增加静态分析来减少动态分析中漏报率较高的问题。

3.4.5 缺少高层语义信息

由于底层的二进制代码缺乏必要的语义和类型信息，限制了动态污点分析技术在二进制程序分析中的应用。诸葛建伟 [2] 等人提出了基于类型感知的动态污点分析技术和面向类型变量的符号执行技术，有效解决了目前工作在二进制层次上的污点分析技术无法为软件漏洞分析提供高层语义的问题。该技术充分利用输入点的类型信息进行类型传递与推导，同时结合 Type Sink 点的类型信息，将二进制程序分析中的动态污点分析提升到变量粒度，且在污点传播的同时进行面向类型变量的符号执行，使得漏洞分析与特征提取具有更好的语义支持。

动态污点分析的优势在于可以通过执行程序得到准确的程序运行时信息，但是由于动态污点分析进行一次动态运行只能执行单一的程序路径，在进行多次执行后，仍然会存在没有覆盖到的路径，那么这些未被覆盖的路径上存在的安全问题就会被忽略，也就存在部分污点信息泄露问题。因此，动态污点分析可以与符号执行技术结合使用，

从而提高路径的覆盖率。目前针对动态污点分析技术的研究重点包括传播逻辑的设计、分析效率的优化及隐式流分析等。对于当前的污点分析工具不能兼顾速度和准确度的不足，一些研究人员提出了一种粗细粒度结合的动态污点分析方法，该方法结合两者的优势通过在线粗粒度模式保证了污点分析信息采集的快速性，同时采用离线细粒度模式以合理的时间消耗提升了污点分析的精确度。未来将根据中间语言获取的信息流进行动态污点分析，并结合制定的安全策略进行漏洞判定，同时也会与混合符号执行技术相结合从而进行漏洞的判定。

4 混合符号执行技术

4.1 符号执行的基本原理和分类

4.1.1 符号执行和混合符号执行

符号执行的基本思想是用抽象符号代替程序变量，或将程序变量的值表示为符号值和常量组成的计算表达式，模拟程序执行，从而进行相关分析 [3]。采用符号执行有以下几点优势：首先，一次符号执行相当于具体执行一组可能无限多个的输入，这组输入可以称为输入等价类；其次，变量之间的代数运算关系能够被发现，这样就使得程序的内在逻辑便于理解；最后，符号执行准确地记录了路径的约束条件，从而便于进一步判断路径可行性以及根据约束条件构造程序的测试用例。

但是，在静态符号执行中由于缺乏程序动态运行时信息，程序执行很难被完全模拟，分析不够准确。为解决这个问题，相关研究人员提出了混合符号执行技术，这是一种动态分析技术。混合符号执行的核心理念是在程序真实运行环境中，判断程序哪部分可以直接运行，哪部分需要经过符号执行。这样程序的运行时信息就被充分利用，提高了分析的准确性。

4.1.2 二进制程序的混合符号执行分类

面向二进制的混合符号执行从执行过程中是否同时进行分析的角度，可以分为在线符号执行和离线符号执行；从分析对象形态的角度，可以分为面向底层指令系统的符号执行和面向中间代码的符号执行 [3]。

在线符号执行是指程序执行和符号计算同时进行，如 SmartFuzz[28]；离线符号执行是指首先记录程序执行轨迹，然后根据轨迹执行的重放进行符号计算，如 SAGE[29]。

在线符号执行面临以下几个问题 [28,29]。1）由于符号执行过程会占用较多资源，在线符号执行可能会降低目标程序的性能。例如，在符号执行过程中，目标程序可能会由于缺乏计算资源而提前退出，这限制了程序执行轨迹的深入，导致最后收集到的程序可能缺少一些必要的内部信息。2）通常程序在执行过程中会存在并发情况及一些不确定事件，这可能会给在线符号执行带来一些不可预料的问题。多线程或多进程程序中原有的同步关系很可能会被在线混合符号执行所破坏，这导致目标程序可能会出现资源冲突甚至错误的情况。3）如果目标程序中采用了代码混淆、加壳等保护技术，混合符号执行很可能无法正常运行。基于上述问题，在未来研究工作中将采用离线符号执行技术进行漏洞挖掘。

采用离线符号执行方式可以有效缓解上述问题 [3]。一方面，在进行离线符号执行的过程中，仅记录程序执行轨迹，无需进行符号计算，这给目标程序带来很小的性能损失；甚至在硬件虚拟化条件支持下，也完全可以在硬件层面实现执行轨迹的记录 [3]。另一方面，仅实现记录程序执行轨迹的功能相对简单，给目标程序执行带来的影响较小，不会破坏目标程序多线程或多进程的同步关系。而且，执行轨迹重放过程中的符号计算完全可以在高性能计算平台上实现，这也意味着可以在与目标程序不同的平台上运行，进而提高符号分析的性能。

面向底层指令系统的符号执行是指根据底层指令的语义进行符号执行，如 SAGE[29]；面向中间代码的符号执行是指将执行轨迹上的底层指令翻译到中间代码，再对中间代码进行符号计算，如 SmartFuzz[28]。

面向中间代码的符号执行的主要优势是对于不同的指令系统，其具有很好的兼容性。同时符号执行引擎是对中间代码进行解释而非底层指令，这使其具有很强的可扩展性。对于在不同指令系统上运行的程序，只需要重新实现执行轨迹记录，能够将不同的指令系统翻译成同一种中间代码，就可对该平台上运行的程序进行相应的分析 [30]。对执行轨迹进行重放的过程中，首先会将底层指令翻译成一种中间语言，再根据中间语言获取相应的信息流，进而进行混合符号执行。

4.2 符号执行的过程

通常情况下，使用符号执行进行漏洞分析的基本过程如图 5 所示。主要包括二进制程序建模和漏洞分析两大部分。

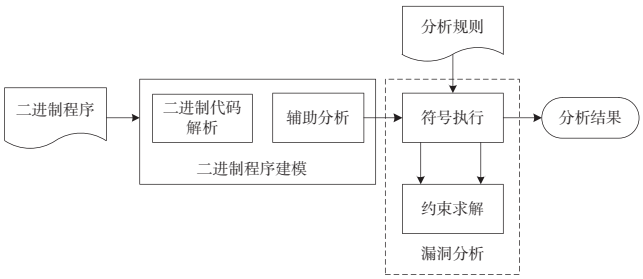

图 5 符号执行的基本过程

1）二进制程序建模

对二进制程序进行建模包括对二进制代码进行基本解析，获得程序代码的中间表示。由于符号执行过程通常是路径敏感的分析过程，因此在二进制代码进行解析之后，系统通常需要构建结构图去描述程序路径，如程序的调用图和控制流图，这些图可能是符号执行分析过程中需要用到的，用以辅助符号执行。

2）漏洞分析

漏洞分析过程主要包括符号执行和约束求解两部分。符号执行将变量的取值表示为符号和常量的计算表达式，并将路径条件和程序存在漏洞的条件表示为符号取值的约束；约束求解过程不仅要判断路径条件是否满足，根据判断结果对路径进行取舍，而且还要检查程序存在漏洞的条件是否满足。在符号执行的过程中，通常需要利用一定的漏洞分析规则，分析规则描述了在什么情况下需要引入符号以及在什么情况下程序可能存在漏洞等信息。

4.3 符号执行的发展及研究现状

符号执行是在 1975 年提出的[31]，最早应用于软件测试领域。后来，BUSH[32] 等人提出了静态符号执行工具 Prefix，采用路径敏感和过程间分析，降低了误报，同时将符号执行技术应用在漏洞挖掘领域。在模拟执行时，为避免路径爆炸，Prefix 在每个函数内部最多只检查 50 条路径，在所有路径结束后，Prefix 合并所有路径的结果，建立函数的摘要，再次调用该函数时可直接应用已构建摘要，加速模拟。之后微软收购了 Prefix，并在此基础上实现了 Prefast，Prefast 已在微软内部成为标准源代码静态检测工具之一。

静态符号执行技术在分析时需要模拟程序执行环境，且由于其缺乏程序动态运行时的相关信息，程序执行很难被完全模拟，分析不够准确。为解决此问题，研究人员提出了混合符号执行的概念，将符号执行从一种程序静态分析技术转变为动态分析技术，并实现了一系列代表工具，如 CUTE[33]、DART[34] 等。CUTE[33] 采用混合执行技术，以内存图表示测试单元输入，在符号执行过程中生成约束表达式并进行追踪。为减少符号化执行带来的开销，CUTE 采取了随机测试和符号化相结合的方式，默认情况下采用随机测试的方法对程序进行检测。当随机测试接近饱和时，才通过符号化执行生成测试用例，然后转向新的路径继续执行。CADAR[35] 等人开发的 KLEE 是使用符号执行技术构造程序测试用例的开源工具，可用于分析 Linux 系统下的 C 语言程序，在分析程序构造测试用例的同时，也利用符号执行和约束求解技术在关键的程序点上对符号的取值范围进行分析，检查符号的取值范围是否在安全规定的范围之内。如果分析中发现符号的取值不能满足安全规定，则认为程序存在相应的漏洞。KLEE 在构造测试用例时，不仅考虑路径条件，也考虑触发程序漏洞的条件。

由于面向源代码的工具会由于无法获得程序源代码而使用受限，因此相关研究人员提出了在二进制代码层面实现混合符号执行技术，并基于此开发出很多工具，如 SAGE[29]、BitBlaze[5]、SmartFuzz[28] 等。SAGE[29] 以合法输入对程序进行动态测试，并借助二进制分析平台 iDNA 追踪和记录指令执行过程，在指令重放中收集路径约束条件并逐一求解，生成新的测试集以驱使程序执行不同路径，最大化代码覆盖率。BitBlaze[5] 是一个二进制分析平台，集成了静态分析、动态分析、动态符号执行等技术，主要由 3 部分组成，即 Vine、TEMU 和 Rudder，分别实现静态分析、动态分析和动态符号执行等功能。SmartFuzz[28] 是基于 Valgrind 二进制插桩平台的工具，可用于发现 Linux 系统下二进制程序中存在的整数漏洞。SmartFuzz 先将二进制代码转换成 VEX 中间语言，再进行在线符号执行和约束求解，并动态生成测试用例。

4.4 存在的问题

在目前的符号执行技术研究中主要存在以下两方面问题。

4.4.1 路径爆炸问题

符号执行技术面临着一个主要的问题，即路径爆炸问题。其主要形成原因是每当遇到一个类似 if 的语句，都有可能会使现在的路径集合在原来的基础上再多一条新的路径。这种路径的增长方式是指数级的。

解决路径爆炸问题主要有以下几种方法：1）对每个过程内路径的数目进行约定限制，这种方法和限界符号执行有些相似。2）通过改进路径调度算法来提高符号执行的分析性能。例如，BOONSTOPPEL[36] 等人根据程序执行的方式及特点，在 EXE 工具的基础上提出了将路径进行裁剪的思想，通过裁剪思想减少了一些不必要的路径分析。此种方法被证明可以很好地缓解符号执行过程中路径爆炸的问题。3）采用符号执行的并行化策略 [35]，根据目前并行计算技术来提高约束求解器的求解速度。

4.4.2 性能问题

符号执行技术会使用约束求解器对路径条件的表达式进行求解，在进行程序分析时，同时还需要对程序进行插桩处理。据统计，2000 行 C 代码插装后能达到 40000 行 [37]，这不仅意味着增加了大量的代码行数，也意味着增加了对求解器的调用次数，而求解器的性能决定了符号执行的效率。目前的解决思路是考虑如何减少求解器的使用频率，主要有两种方法 [29,35]，一是借助缓存技术将常用的约束表达式或者求解过程中得到的部分中间表达式进行缓存，从而减少一些不必要的重复求解；二是通过进一步优化约束条件来提高求解速度，如消去无关约束、重写表达式及简化约束集合等。

如今，混合符号执行已经成为重要的代码执行空间遍历技术。与传统的静态符号执行技术不同，混合符号执行技术充分利用了程序运行时的信息，提高了符号执行的准确性，促进了符号执行的发展。此外，随着业界对二进制程序分析需求的不断增长，面向二进制程序的混合符号执行技术变得越来越重要。未来的主要研究方向有以下两个方面：

1）可以进一步将符号执行与其他技术相结合。具体来说，充分利用现有技术的优势与符号执行技术进行优势互补，从而弥补单个分析技术所面临的一些问题。

2）对于符号执行中所面临的路径爆炸问题，还需要进一步改善。可以采用抽象的思想，将众多的路径信息进行处理、抽象，从而形成一个更高层次的集合，这样就只需要对这些抽象集合进行遍历即可，从而大大减少了路径的数量，可以缓解路径爆炸问题。

5 模糊测试技术

5.1 模糊测试技术的基本原理和分类

模糊测试技术是软件漏洞分析的代表性技术，最早是由 MILLER[38] 等人在 1989 年提出的。由于其不需要对目标程序进行深入的分析，而且通过模糊测试发现的漏洞一般是真实存在的，因此模糊测试技术在软件漏洞分析领域占据重要地位。模糊测试的基本思想是：向待测程序提供大量特殊构造的或是随机的数据作为输入，监视程序运行过程中的异常并记录导致异常的输入数据，辅以人工分析，基于导致异常的输入数据进一步定位软件中漏洞的位置 [1]。从测试用例生成策略的角度，模糊测试可以分为基于变异和基于生成的模糊测试；从代码的开放性角度，模糊测试可以分为黑盒模糊测试和白盒模糊测试。

基于变异的模糊测试方法采用对已有的数据进行变异的技术创建新的测试用例；基于生成的模糊测试方法则是通过对具体的待测程序进行建模，从头开始产生测试用例 [1]。

黑盒模糊测试最初被称为随机测试，是典型的黑盒动态分析技术。其主要思想是向被测程序输入大量不规则的数据来检测系统是否发生不正常的行为。这种方法具有简单、执行较快的特点，但本身的缺陷是测试用例的生成具有盲目性，即缺乏针对性，因此路径覆盖率较低。通常所说的模糊测试一般是指黑盒模糊测试。

白盒模糊测试是近年来逐渐流行的另一种模糊测试技术，其综合了模糊测试和符号执行。其基本思想是：首先符号执行技术对程序的路径进行遍历，并进行路径条件分析，对路径条件进行求解，得出路径是否可达的信息；然后针对可达的路径，根据路径约束条件生成特定的测试用例，从而进行模糊测试。这种方法由于结合符号执行技术辅助生成测试用例，因此其路径覆盖率高。但同时也因为该方法结合了符号执行，从而引入了路径爆炸以及约束求解能力有限的问题。

5.2 模糊测试的基本过程

模糊测试在执行的过程中，大致经历 5 个基本阶段 [1]，如图 6 所示。

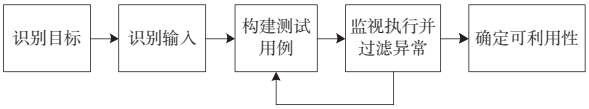

图 6 模糊测试的基本过程

1）识别目标。为了选择合适的模糊测试工具或技术，首先需要充分了解目标程序，明确目标程序。

2）识别输入。通常由于程序在接收外部输入时未能正确处理一些非法数据而导致大多数漏洞被利用，因此考虑程序所有可能的输入情况对模糊测试的成功具有重要指导意义。

3）构建测试用例。构建测试用例是模糊测试过程中较为关键的一步，测试用例的生成策略决定了模糊测试的测试效率。测试用例生成方法可以分为两大类：基于变异和基于生成的测试用例生成策略。测试用例集的构建可以采用其中的一种策略，也可以同时采用上述两种策略。常见的测试用例构建方法有 3 种：随机生成测试用例、预先生成测试用例、变异或强制生成测试用例。

4）监视执行并过滤异常。根据测试用例在目标程序中的执行情况，观察目标程序是否出现异常情况或者系统崩溃，并根据异常结果进行漏洞判定。

5）确定可利用性。根据应用程序的不同对已经识别的漏洞进一步判定，确定此漏洞是否可以被利用。

5.3 模糊测试的发展及研究现状

模糊测试很早就在软件工程中被采用，最初被称为随机测试，它起初的关注点并不是评价软件的安全性，而是用来验证代码的质量 [1]。1989 年，MILLER[38] 等人开发了模糊测试工具，用来检测 UNIX 系统上程序的健壮性。这种早期的模糊测试方法是一种较为简单的黑盒测试：简单地构造随机字符串并传递给目标应用程序，如果应用程序在执行过程中发生异常，那么就认为测试失败，否则就认为通过，结果发现该方法使得程序崩溃的概率在 25% 左右。1999 年，芬兰奥卢大学的协议测试项目组开发了网络安全测试软件 PROTOS，并首次提出了测试集的概念 [39]。

2002 年，AITEL[40] 发布了模糊测试框架 SPIKE。该框架专门用于测试网络协议，采用基于块的方法，对于测试包含变量和相应变量长度的数据块，效果显著，同时允许用户创建自己定制的模糊测试器。2004 年，EDDINGTON[41] 发布了一款同时支持网络协议和文件格式的模糊测试框架 Peach。Peach 使用 Python 进行编写，主要使用一个 XML 文件指导整个模糊测试的过程，同时因其支持跨平台操作，因此几乎可以检测任何的网络协议。2007 年，AMINI[42] 发布了模糊测试框架 Sulley。Sulley 是一个基于 Python 实现的模糊测试框架，主要对各种网络协议进行模糊测试，是由多个可扩展的组件构成，包括数据生成、代理和工具等。Sulley 的优势在于使用简单，并且是基于路径的分析，考虑了路径覆盖的问题。

在模糊测试技术应用于漏洞挖掘方面，一些学者也相应地研发出一些工具。针对模糊测试在遇到校验和防护机制时存在的局限性，2010 年，北京大学的 WANG[43] 等人结合混合符号执行和细粒度动态污点传播技术，提出了一种可以绕过校验和机制的方法，并以此开发了对应的工具 TaintScope，为应用模糊测试发现深层次的漏洞扫除了障碍。2011 年以后，遗传算法以及启发式算法等被很多研究学者用于辅助生成测试用例，从而提高了测试用例的生成效率 [42]。

5.4 存在的问题

模糊测试相比其他漏洞挖掘方法有很多优势，但也有不足之处，仍然存在很多局限性，其中比较重要的是模糊测试的测试用例并不能保证覆盖到所有的语句分支，模糊测试的自动化程度目前还不高。因此，优化模糊测试的测试用例生成，提高模糊测试的效率仍然是主要的研究方向。

5.4.1 测试用例生成策略有待改进

测试用例生成是模糊测试中最关键的部分，它的效率关系到整个模糊测试的效率。早期的模糊测试往往使用盲目枚举的测试用例生成策略，属于黑盒测试方法。使用这种测试方式，虽然测试用例的规模可能已经达到了很高的数量级别，但是其测试效果往往并不显著。而且在一些软件中，广泛应用私有数据和校验和，经常因为不能通过校验和而将这些随机产生的数据直接丢弃，在较深层次的漏洞难以检测，以至于测试的效果并不理想。因此，如何优化测试用例的生成策略是当前需要重点研究的问题。

目前，有研究人员将基因学中的一些相似的概念引入

到测试用例的生成策略中。例如，选择和交叉重组等概念，并对测试用例的生成策略进行优化。其中，使用遗传算法辅助生成特定的测试用例 [44] 已被证明在很大程度上提高了测试用例的生成效率，是一种很有用的自动化测试技术。

同时，也可以考虑将模糊测试与其他二进制漏洞分析技术相结合。例如，动态污点分析能够提取程序的动态数据流或控制流信息，并融合到二进制漏洞分析过程中，指导模糊测试中输入数据的生成，使传统的黑盒模糊测试能够有的放矢，极大地改进了模糊测试的效果。

5.4.2 自动化程度有待提高

模糊测试是一种盲注入的方式，手工测试需要花费较多时间，导致了它的实用价值受到影响。在常见的一些模糊测试框架中，经常需要人工参与确定输入数据的约束以及生成测试用例等。因此目前研究的关注点大都在于自动化与智能化的模糊测试方法。一种解决思路仍然是考虑将基因学中的遗传算法用于其中，从而提高测试的自动化程度。

5.4.3 无法解决多点触发的漏洞

模糊测试技术经常只能发现由一个条件引起的漏洞，而没有办法发现需要多个条件才能引发的漏洞 [45]。近年来，一些研究人员在挖掘多点触发漏洞方面试图利用模糊测试，并提出了多维模糊测试的概念 [46]。然而，多维模糊测试目前存在组合路径爆炸的问题，因此发展缓慢，还需要进一步研究。

相比其他漏洞挖掘方法，模糊测试具有很大的优势，由于它不需要了解程序的内部结构，因此开销较小。与此同时，模糊测试也存在一些局限性，如测试盲目性较大、测试效率较低及代码覆盖率无法保证等问题。为了解决这些问题，可以考虑将模糊测试技术与其他二进制漏洞分析技术相结合，如与动态污点分析或者符号执行结合。在未来的研究工作中，会将模糊测试集成到构建的二进制分析框架中，将其与符号执行技术相结合。由于符号执行技术具有很好的路径覆盖率，因此在生成测试用例时，可以先利用符号执行技术对程序进行约束求解，得出路径约束集合，再根据路径的约束条件有针对性地设计测试用例，进而执行模糊测试。模糊测试与符号执行结合生成有导向的测试用例在理论上会大大提高模糊测试的效率，检测效果也会明显提高。除此之外，使用基因学中的遗传算法来辅助生成测试用例也是一个比较好的途径。鉴于此方法，将来也可以进一步关注其他学科领域中的一些类似的概念和思想，判断其能否应用于漏洞分析领域，解决目前漏洞分析技术所面临的一些问题，这也是一个新的突破点。

6 结束语

漏洞分析本身是一项较为复杂的课题，国际上的研究历史并不长，国内的相关研究工作也刚刚起步。未来的发展方向将主要在于结合各种漏洞分析技术，取长补短，以此来弥补单种漏洞分析技术所面临的不足，同时研究并行化、形成规模化的漏洞分析也是一个比较重要的研究方向。

本文以面向二进制代码的漏洞挖掘技术为调研点，通过对目前较为典型的二进制漏洞分析框架的研究，提出了在未来工作中对二进制程序漏洞挖掘技术研究的整体思路，进一步构建一个更为强大的综合的二进制漏洞分析框架，在此基础上集成一些具有代表性的分析技术，并对这些技术进行优化。通过对这些关键点以及关键技术进行调研，了解了这些关键技术的研究现状，并总结出这些关键技术目前所面临的一些问题。未来，将会对这些关键技术进一步优化，并集成在将要构建的二进制分析平台中，从而形成一套完整的、规模化的二进制漏洞分析框架。（责编 潘海洋）

参考文献：

[1] 吴世忠，郭涛，董国伟，等 . 软件漏洞分析技术 [M]. 北京：科学出版社，2014.

[2] 诸葛建伟，陈力波，田繁，等 . 基于类型的动态污点分析 [J]. 清华大学学报：自然科学版，2012，52（10）：1320-1321.

[3] 王铁磊 . 面向二进制程序的漏洞挖掘关键技术研究 [D]. 北京：北京大学，2011.

[4] 姜玲燕，梁阿磊，管海兵 . 动态二进制翻译中的中间表示 [J]. 计算机工程，2009，35（9）：283-285.

[5]SONG D, BRUMLEY D, YIN Heng, et al. BitBlaze: A New Approach to Computer Security via Binary Analysis[C]//Springer. 4th International Conference on Information Systems Security, December 16-20, 2008, Hyderabad, India. Berlin: Heidelberg, 2008: 1-25.

[6]SHOSHITAISHVILI Y, WANG Ruoyu, SALLS C, et al. The Art of War: Offensive Techniques in Binary Analysis[C]//IEEE. 2016 IEEE Symposium on Security and Privacy, May 22-26, 2016, San Jose, CA, USA. New Jersey: IEEE, 2016: 138-157.

[7] 陈凯明，刘宗田 . 符号执行过程的 DFA 和 CFA[J]. 计算机工程，2002，28（11）：95-96.

[8]CIFUENTES C, EMMERIK V M. UQBT: Adaptable Binary

Translation at Low Cost[J]. Computer, 2000, 33(3): 60-66.
[9]BRUSCHI D, MARTIGNONI L, MONGA M. Boomerang[EB/OL]. http://boomerang.sourceforge.net/index.php, 2017-2-11.
[10] 马金鑫，李舟军，忽朝俭，等 . 一种重构二进制代码中类型抽象的方法 [J]. 计算机研究与发展，2013，50（11）：2418-2428.
[11]NETHERCOTE N, SEWARD J. Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation[J]. ACM Sigplan Notices, 2007, 42(6): 89-100.
[12]DENNING D E. A Lattice Model of Secure Information Flow[J]. Communications of the ACM, 1976, 19(5): 236-243.
[13] 黄强，曾庆凯 . 基于信息流策略的污点分析传播分析及动态验证 [J]. 软件学报，2011，22（9）：2036-2048.
[14] 史大伟，袁天伟 . 一种粗细粒度结合的动态污点分析方法 [J]. 计算机工程，2014，40（3）：12-17.
[15] 代伟，刘智，刘益和 . 基于二进制代码的动态污点分析 [J]. 计算机应用与研究，2014，31（8）：2497-2505.
[16]YIN Heng, SONG D. Temu: Binary Code Analysis via Whole-system Layered Annotative Execution[EB/OL]. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-3.pdf, 2016-6-11.
[17]KANG M G, MCCAMANT S, POOSANKAM P, et al. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation[EB/OL]. https://people.eecs.berkeley.edu/~dawnsong/papers/2011%20dta++-ndss11.pdf, 2011-2-6.
[18] 马金鑫，李舟军，张涛，等 . 基于执行踪迹离线索引的污点分析方法 [EB/OL]. http://www.jos.org.cn/ch/reader/create_pdf.aspx?file_no=5179&journal_id=jos, 2017-2-20.
[19] 王蕾，李丰，李炼，等 . 污点分析技术的原理和实践应用 [J]. 软件学报，2017，28（4）：1-11.
[20] 宋铮，王永剑，金波，等 . 二进制程序动态污点分析技术研究综述 [J]. 信息网络安全，2016（3）：77-83.
[21]NEWSOME J, SONG D. Dynamic Taint Analysis for Automatic Detection, Analysis and Signature Generation of Exploits on Commodity Software[J] Chinese Journal of Engineering Mathematics, 2005, 29(5): 720-724.
[22]CLAUSE J, LI Wanchun, ORSO A. Dytan: A Generic Dynamic Taint Analysis Framework[C]//ACM. 2007 International Symposium on Software Testing and Analysis, July 9-12, 2007, London, United Kingdom. New York: ACM, 2007: 196-206.
[23]QIN Feng, WANG Cheng, LI Zhenmin, et al. Lift: A Low-overhead Practical Information Flow Tracking System for Detecting Security Attacks[C]//IEEE. 39th Annual IEEE/ACM International Symposium on Microarchitecture, Orlando, FL, USA, December 9-13, 2006. New Jersey: IEEE, 2006: 135-148.
[24] 彭建山，奚琪，王清贤 . 二进制程序整型溢出漏洞的自动验证方法 [J]. 信息网络安全，2017（5）：14-21.
[25] 林伟，蔡瑞杰，祝跃飞，等 . 基于语义规则的污点传播分析优化方法 [J]. 计算机应用，2014，34（12）：3511-3514.
[26] 朱正欣，曾凡平，黄心依 . 二进制程序的动态符号化污点分析 [J]. 计算机科学，2016，43（2）：155-158.
[27] 崔化良，兰芸，崔宝江 . 动态污点分析技术在 ActiveX 控件漏洞挖掘上的应用 [J]. 信息网络安全，2013（12）：16-19.
[28]MOLNAR D, LI Xuecong, WAGNER D. Dynamic Test Generation to Find Integer Bugs in x86 Binary Linux Programs[C]//ACM. 18th Conference on USENIX Security Symposium, August 10-14, 2009, Montreal, Canada. New York: ACM, 2009: 67-82.
[29]GODEFROID P, LEVIN M Y, MOLNAR D A. Automated Whitebox Fuzz Testing[EB/OL]. https://www.researchgate.net/publication/221655409_Automated_Whitebox_Fuzz_Testing, 2017-2-12.
[30] 陆萍萍 . 二进制代码的漏洞挖掘技术研究 [D]. 北京：北京理工大学，2013.
[31]BOYER R S, ELSPAS B, LEVITT K N. SELECT—A Formal System for Testing and Debugging Programs by Symbolic Execution[J]. ACM SigPlan Notices, 1975, 10(6): 234-245.
[32]BUSH W R, PINCUS J D, SIELAFF D J. A Static Analyzer for Finding Dynamic Programming Errors[J]. Software-Practice and Experience, 2000, 30(7): 775-802.
[33]SEN K, MARINOV D, AGHA G. CUTE: A Concolic Unit Testing Engine for C[J]. ACM Sigsoft Software Engineering Notes, 2005, 30(5): 263-272.
[34]GODEFROID P, KLARLUND N, SEN K. DART: Directed Automated Random Testing[EB/OL]. http://people.cs.vt.edu/~ryder/6304/lectures/12-DART_Godefried_PLDI2005_MarkusK-slides.pdf, 2017-2-11.
[35]CADAR C, DUNBAR D, ENGLER D R. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs[C]//ACM. 8th USENIX Symposium on Operating Systems Design and Implementation, December 8-10, 2008, San Diego, California. New Jersey: 2008: 209-224.
[36]BOONSTOPPEL P, CADAR C, ENGLER D. RWset: Attacking Path Explosion in Constraint-based Test Generation[C]//Springer. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, March 29-April 6, 2008, Budapest, Hungary. Berlin: Heidelberg, 2008: 351-366.
[37] 李舟军，张俊贤，廖湘科，等 . 软件安全漏洞检测技术 [J]. 计算机学报，2015，38（4）：717-728.
[38]MILLER B P, FREDRIKSEN L, SO B. An Empirical Study of the Reliability of UNIX Utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.
[39]KAKSONEN R. A Functional Method for Assessing Protocol Implementation Security[EB/OL]. http://www.vtt.fi/inf/pdf/publications/2001/p448.pdf, 2017-2-15.
[40]AITEL D. An Introduction to SPIKE, the Fuzzer Creation Kit[EB/OL]. https://wenku.baidu.com/view/54baf59e51e79b89680226d6.html, 2017-2-15.
[41]EDDINGTON M. Peach[EB/OL]. http://peachfuzzer.com, 2017-2-10.
[42]AMINI P. Sulley[EB/OL]. http://code.google.com/p/sulley, 2017-2-10.
[43]WANG Tielei, WEI Tao, GU Guofei, et al. TaintScope: A Checksum-aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection[C]//IEEE. 2010 IEEE symposium on Security and Privacy, May 16-19, 2010, Oakland, CA, USA. New Jersey: IEEE, 2010: 497-512.
[44] 杜晓军，林柏钢，林志远，等 . 安全软件模糊测试中多种群遗传算法的研究 [J]. 山东大学学报：理学版，2013（7）：79-84.
[45] 史记，曾昭龙，杨从保，等 . Fuzzing 测试技术综述 [J]. 信息网络安全，2014（3）：87-91.
[46] 王连赢 . 文件触发类二进制程序漏洞挖掘技术研究 [D]. 北京：北京邮电大学，2015.