二进制程序安全缺陷静态分析方法的研究综述

田 硕 梁洪亮
(中国科学院软件研究所 北京 100190)

摘 要 对现有二进制程序安全缺陷静态分析方法进行了综述和分析,提出了整个程序分析过程中的关键问题以及二进制程序安全分析的主要研究方向。通过对二进制程序缺陷静态分析流程的总结,发现二进制程序信息恢复是整个分析过程的关键,构造内容丰富的、通用的中间表示是二进制程序缺陷分析的重要研究方向。

关键词 二进制程序,静态分析,安全缺陷,中间表示,反汇编
中图法分类号 TP312 **文献标识码** A

Survey of Static Analysis Methods for Binary Code Vulnerability

TIAN Shuo LIANG Hong-liang
(Institution of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract A survey of static analysis methods for binary code vulnerabilities was provided. Based on summing up existing static detect methods for vulnerabilities, the general procedural of binary static analysis was modeled, and transforming program information into expressive and generic intermediate representation was regarded as the key component of the analysis and as a important research direction.

Keywords Binary code, Static analysis, Security vulnerability, Intermediate representation, Disassemble

1 简介

目前,在市场作用力下,大部分的商业软件以二进制程序的形式发布,很多软件产品根据客户需求进行外包开发。很多没有源码的应用程序被部署到计算机系统,即使在开源系统中也不可避免地要用到二进制程序。如何确定这些二进制程序中是否存在安全缺陷是急需解决的一个问题。

程序安全性是指程序执行的指令动作不会违背系统的安全策略。源程序经过编译、优化等处理得到可执行程序,源程序的语义可能与程序的实际执行动作不同[1]。因此,源程序缺陷检测并不能保证程序的安全性。可执行程序以0/1二进制字节流表示机器指令,直接表示程序的执行动作。

目前市场上的二进制程序安全检查工具主要有两种:HBGary公司的BugScan[2]和@Stake公司(被赛门铁克收购)的SmartRisk Analyzer[3]。BugScan是一种跨平台的硬件软件结合的二进制程序缺陷检测工具,其检测过程对用户是完全透明的。SmartRisk Analyzer是Windows平台下的静态二进制程序分析工具,除了检测二进制程序,它还监控程序载入内存时调用的动态链接库和配置环境,检测第三方插件的安全性。虽然这两种工具能够对程序进行"深度静态分析",但是由于SmartRisk Analyzer价格昂贵,中小型公司机构难以承担,市场占有率不高,BugScan用户经验表明,程序分析过程中仍存在一些问题[4,5]。所以,这两种工具都不能满足日益增长的程序安全性需求。

针对二进制程序易执行的特点,很多二进制程序安全分析方法使用动态分析技术[6,7]。动态分析基于程序监控,发现程序执行中的不安全行为。与之相对,静态分析方法并不实际执行程序,而是遍历程序代码分析程序的属性。相比之下静态分析方法具有以下三点优势[8]:

1. 静态技术可进行全面分析。静态分析方法能够保证程序的所有执行路径得到检测,而不局限于特定的执行路径。

2. 静态分析可以在执行之前验证程序的安全性,进而在运行程序前检验程序安全性、修补程序安全漏洞。

3. 静态分析无需实际执行被检测程序,不会产生程序运行开销。

本文主要介绍二进制程序静态分析方法的研究现状,第2节简单说明二进制程序的特点以及程序安全性分析的过程,第3节介绍目前主要的分析方法,最后是总结研究现状并展望二进制程序安全分析的研究方向。

2 二进制程序安全分析简介

2.1 二进制程序的结构和特点

可执行程序生成过程如图1(a)所示,高级语言程序(或者汇编程序)经过编译(汇编)处理生成的二进制目标文件,相关目标文件经过连接器(linker)连接得到可执行程序。程序运行时,可执行程序被加载到内存并开始执行。二进制可执

到稿日期:2008-08-25 返修日期:2008-12-29 本文受863计划重点项目(2007AA010601),国家自然科学基金(60673022),北京市科委项目(Z08000102000801)资助。

田 硕(1986-),女,硕博研究生,主要研究方向为程序安全性分析等,E-mail:tianshuo9@gmail.com;梁洪亮(1976-),男,副研究员,硕士生导师,主要研究方向为系统软件关键核心技术、基础软件高可信技术、移动系统及其安全理论与方法等。

行程序有如下特点：

1. 难读性。二进制程序的内容是二进制字节流。

2. 易执行。二进制程序以二进制的形式表示的 CPU 指令，可以直接在目标机器上运行。

3. 完备性。二进制程序具有自我完备性，包括了执行所需要的库文件。

4. 可移植性。可执行软件能够跨平台运行。

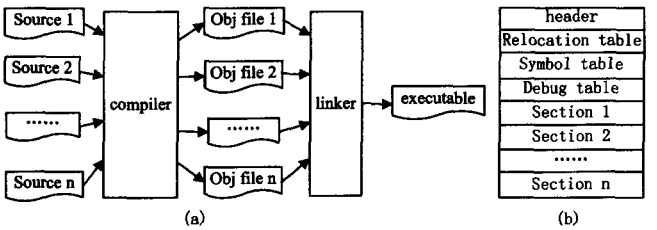

图 1

由于以上特性，二进制程序被作为软件的主要载体形式，尤其是难读性有助商业软件保护产权。

如图 1(b)所示，目标文件一般包括 5 个基本部分：

1. 文件头。文件的总体信息，例如文件长度、创建日期等。

2. 重定位信息。目标文件加载到内存地址时所用的一组地址。

3. 符号表。供连接器使用，相关目标文件连接成可执行程序时所用的一组全局符号。

4. 调试信息。程序调试过程中用到的信息，例如代码行号、变量名等。

5. 代码和数据。二进制指令以及源码产生的数据，多以划分(section)的形式存放。

二进制可执行程序有多种格式，常用的有 MS-DOS 的 .COM 文件、Unix 的 a.out 文件、ELF 文件和 Windows 的 PE 文件格式等。

2.2 二进制程序安全分析评测标准

二进制程序安全性分析的两个主要评测指标：覆盖率和精确度。覆盖率是指程序分析的完整性，覆盖率越高，缺陷检测过程中缺陷发现的漏报率越低。精确度是指分析结果的正确性，精确度越高，缺陷检测过程中缺陷发现的误报率越低。静态分析全面检查程序的所有可能行为，不同于动态分析监控程序具体执行动作，能够有效地提高程序分析的覆盖率，降低缺陷检测的漏报率。随着覆盖率提高，静态分析方法带来了精确度降低的问题。因此应当权衡提高精确度、排除缺陷误报的开销和提高覆盖率、提高缺陷漏报的收益，使分析方法在缺陷发现过程中达到最优效果。

2.3 二进制程序安全分析过程

二进制程序静态安全分析方法分为两种：基于程序结构的分析过程和基于程序语义的分析过程。基于程序结构的分析根据程序格式，检查程序各部分内容，发现可能存在的安全威胁，具体分析方法见 3.1 节基于程序结构的静态分析方法。基于程序语义的分析过程结合程序信息，发现程序执行语义中的不安全动作，这种分析过程在精确度和覆盖率方面都要优于前者。模型检测、符号执行和程序注释是基于程序语义的安全分析常用的方法。

基于程序语义的二进制程序静态分析过程大致分为两个阶段(如图 2 所示)：程序信息恢复和安全缺陷检测。程序信息恢复阶段根据二进制程序信息构建中间表示，缺陷检测阶段使用各种分析方法检测中间表示，发现程序中的安全缺陷。

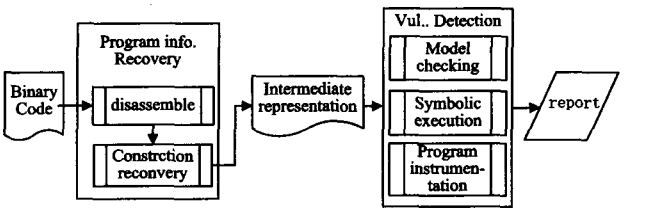

图 2 二进制程序静态分析过程

程序信息恢复是整个分析过程的基础，是程序分析的关键。程序信息恢复即反汇编二进制程序恢复程序控制结构，符号表示程序指令。和源程序相比，二进制程序缺少变量名和变量类型信息，并且无法直接获得控制结构和数据依赖。原始变量名不会引起安全缺陷，变量类型信息可以用变量的长度信息代替。恢复程序的控制结构，获得数据依赖信息是程序信息恢复的重点工作。另外，二进制程序能够直接得到寄存器信息和选中指令。这是二进制程序安全分析的优势，通过寄存器信息和选中指令能够准确检测程序的真实执行动作。而源程序经过编译、优化所得可执行程序的实际执行动作可能与源码语义不完全一致，造成安全性分析不够精确。

中间表示是二进制程序分析的研究重点。合适的中间表示应该具有完整性、通用性。完整性是指能够正确、全面地表示程序信息，通用性则说明中间表示既能用于表示二进制程序信息，同样能够表示源程序信息。二进制程序和源程序采用通用的中间表示，优势在于发展相对成熟的源程序分析技术能够直接用于二进制程序安全分析。WiSA 项目[9]的 T. Reps 等人正在致力于中间表示的研究。

在缺陷检测阶段，使用静态分析方法检测程序安全缺陷，具有无需实际执行程序、系统开销小、漏报率低的优点。但是由于静态分析的特性，会产生较高的误报率，降低误报率是优化静态分析的一个目标。另外可以结合动态分析方法，提高安全分析的精确度和误报率。

3 二进制程序静态分析方法

3.1 基于程序结构的静态分析方法

基于程序格式化的缺陷检测方法无需恢复程序的执行语义，结合文件格式特征静态扫描程序各部分发现安全威胁。Jay-Evan J. Tevis 等人利用这种简单的二进制程序分析方法检测 PE 格式程序的安全性[10,11]，另外，这种方法能够辅助其他二进制程序检测过程[12]。

Jay-Evan J. Tevis 等人研制了一种无需反汇编的 Windows XP/NT 可执行程序缺陷检测方法[10,11]。他们利用可执行程序的文件头、分区和表信息，查看二进制文件内容，可以检测程序中某些异常以及软件安全缺陷。

文献[10,11]针对 PE 格式文件，主要在 3 个方面检测文件安全缺陷：

(1)若一个划分同时存在可写和可执行特性，则程序被加载到内存后，存在执行的过程中被修改的安全缺陷。

(2)若文件中未使用区域补零的数目超过某阈值，则存在该区域被填充恶意代码的安全缺陷。

(3)程序运行过程中调用的 C 库函数和 DLL 文件可能存

在缓冲区溢出等安全缺陷。

基于以上3点，结合PE文件格式完成对程序内容的检查。这种方法无需反汇编程序，就可以发现PE程序中的安全缺陷以及被篡改过的内容。同样，对于其他格式的可执行文件，经过对其结构和内容的分析，静态扫描文件内容也可以对其进行初步的安全漏洞检测。

另外，文献[12]在检测foreign code的过程中，根据二进制程序格式采取一些安全防护手段。例如，利用PE格式程序中导出表(export table)无需修改的特性，在程序载入的过程中将.edata分区设为只读。IAT(Import Address Table)相应的.idata分区仅需要从其他模块中导入其他函数时候需要重写一次，因此.idata初始标记为可读写，而一旦使用加载器修改过一次之后，即被标记为只读。这样就防止了修改.edata和.idata分区的恶意操作。

以上方法依赖于文件头、分区和表等信息，但这些信息的可信度不够。一方面，用于提取文件内容的信息可能不够全面，例如符号表或者导入表可能没有完全包括程序调用所有的C库函数或者DLL文件，文件检查不够全面，不能通过检查文件内容发现所有的安全缺陷。另一方面，检测的对象没有经过安全性分析，这些二进制文件不安全，其内容包括文件表信息等可能已经被恶意篡改。使用被恶意篡改的信息进行安全分析，是没有意义的。因此，这种直接对二进制文件进行静态分析的方法只能粗略地对程序安全性进行分析，其分析结果的覆盖率和精确度都无法保证。

3.2 基于程序语义的静态分析方法

基于程序语义的二进制程序安全分析方法必须首先恢复程序信息。研究者在恢复程序信息方面已经做了大量工作，研究出多种反汇编方法，现从精确度、覆盖率以及性能开销方面，对各种反汇编方法进行比较，如表1所列。

表1 各种反汇编方法精确度、覆盖率以及开销

Parse method	Coverage	Precision	Processing overhead
IDA	35%	100%	Static analysis, overhead 0
Objdump (Linear Sweep)	100%	Very low(complex program)	Static analysis, overhead 0
Recursive transversal	6%～36%	100%	Static analysis, overhead 0
Orso	90%	89%	The overhead is linear in the number of code lines
BIRD	69%～96%	100%	The overhead without initial is lower than 5%

目前公认最强大的商业反汇编工具IDA Pro[13]被广泛应用，并且用于二进制程序分析工具，例如文献[13,14]所述。但是IDA Pro反汇编覆盖率过低，在保证精确性的条件下，仅反汇编确定为代码的指令，造成二进制程序解析不够完整。另外，Linux系统的Objdump命令使用线性扫描的方法反汇编，但无法区分数据和代码，解析精确度低，仅能处理简单的二进制程序。

由于代码中夹杂数据，简单使用线性扫描无法准确的解析指令，需要递归遍历方法辅助。实验表明，单纯地使用递归遍历方法解析二进制代码，其覆盖率是甚至低于1%，即使扩展的递归遍历方法覆盖率仅为6%～36%[15]。文献[16]中结合使用线性扫描和递归遍历，当两种解析的结果相同时，才接受该解析结果。这种方法能够正确解析99.8%以上的代码段指令，其运行时间几乎是单纯的线性扫描方法解析时间和递归遍历方法解析时间之和。

Orso等人研究了一种应对Linn和Debray的迷惑技术的反汇编方法[17]，使用CFG检测验证反汇编的正确性。这种方法的实验结果同Objdump，Linn/Debray的二进制代码解析方法以及IDA Pro的分析结果比对，具有明显优势，覆盖率平均达到了90%。这种方法执行速度快，处理时间和被解析二进制代码的数量成线性关系。

BIRD[15]是一种保证了精确度100%的二进制代码解析方法。这种方法保守的使用递归遍历从二进制程序的入口点静态解析所有直接可达的“正确指令”，对于其他的“可能指令”，使用积分规则判断其正确性。使用源程序通过VC6.0编译得到的包含PDB(Program DataBase)的汇编中间表示为参照，二进制程序代码解析的指令正确性达到100%。但是，和上两种方法比较，覆盖率仅为69%～96%，需要通过进一步改进其中的动态分析方法增大覆盖率。

二进制程序反汇编存在很多困难。其根源在于冯诺依曼系统结构，数据和代码没有区分，反汇编过程中无法区分数据和代码。静态反汇编过程中，间接跳转的目标地址的确定是反汇编要解决的一个关键问题，是决定中间表示的精确度和覆盖率的关键。动态反汇编能够准确解析程序的执行语义，但是覆盖的执行路径仅为整个程序可执行路径的子集。另外，代码迷糊技术更加大了二进制程序解析的难度。

通过二进制代码解析恢复程序信息，构建中间表示，然后就可以根据程序语义静态发现安全缺陷。基于程序语义的二进制程序静态检测常用的技术有模型检测、符号执行和程序注释3种。

3.2.1 基于模型检测的缺陷检测

模型检测通过复杂的模式匹配技术，查看程序执行与预计执行动作(通常表示为规则)是否保持一致，是程序分析常用的方法。模型检测算法遍历程序的状态空间，检测程序执行是否会进入错误状态。模型创建是模型检测的关键和难点，传统的模型检测方法多用于验证网络和通讯协议，模型创建过程复杂，需要大量的人工操作，可能引入人为错误[18]，影响安全分析的准确性。基于规则的模式匹配是一种简化的模型检测方法，根据缺陷类型制定规则，通过模式匹配在程序中发现指定类型缺陷，这种方法常应用于实际的缺陷检测。开源编译器mygcc[19]就是在程序编译过程中模式匹配程序缺陷的抽象语法树表示，实现了GCC基础上的安全缺陷检测功能。

基于模型检测的二进制程序安全缺陷检测需要两个步骤：1)恢复程序信息抽象检测模型；2)发现程序模型中的安全缺陷。如文献[20]根据rootkit的行为制定规则，通过IDA获得程序信息，在符号执行过程中查找符合规则的执行动作，即发现二进制程序中的rootkit缺陷。文献[8]是采用自动机查找模型的检测方法，首先抽象程序得到关键API调用序列，把安全策略用自动机表示，在策略自动机中检测调用序列，查看自动机状态是否进入危险状态，发现程序中的恶意代码。以上这些检测方法仅适用于简单二进制程序中特定类型安全缺陷检测，Wisconsin大学WiSA项目的二进制程序安全分析平台能广泛用于各种复杂二进制程序安全检测[9,21,22]。

Wisconsin 大学的 WiSA 项目结合静态分析将模型检测技术应用于二进制程序安全分析。这种分析方法通过恢复程序中间表示(IR),使用 WPDS 系统抽象表示程序模型。以 WiSA 项目研制的工具 CodeSurfer/x86[21] 为例,介绍模型检测技术在二进制程序安全分析方面的应用。中间表示描述将 Intel x86 可执行程序转化为中间表示形式的过程,3.2.2 节介绍用于安全性分析的 WPDS 系统。

3.2.1.1 中间表示

CodeSurfer/x86[21] 是将二进制程序转化为中间表示的工具。该中间表示与众不同之处在于除了用 CFG 图和过程调用图表示程序控制结构外,它使用 a-locs(abstract-location)结构表示内存使用状态。这种中间表示能够通用于二进制程序表示和源程序表示。整个恢复过程如图 3 所示。

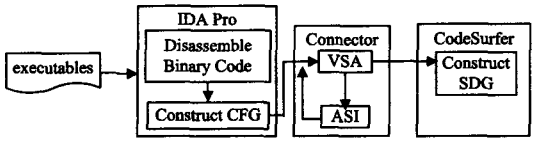

图 3 CodeSurfer/x86 二进制程序信息恢复过程

IDA Pro[13] 是功能最强大的商业反汇编工具,能够识别部分变量、函数边界和库函数调用。CodeSurfer[23] 是一种能够访问程序系统依赖图(SDG)的代码理解和代码检查工具。CodeSurfer/x86 基于这两种工具完成中间表示构造,IDA Pro 提供程序初步反汇编结果,CodeSurfer 根据程序信息创建程序过程间控制流程图(ICFG)和系统依赖图。

IDA Pro 利用静态确定的内存地址和栈空间的偏移地址确定程序变量,得到的程序内存访问和变量信息不完整。另外,由于 IDA Pro 不能处理间接跳转和间接调用,构建的 CFG 和过程调用图不完整。CodeSurfer 需要更加完整、精确的程序信息完成中间表示构造。于是,通过插件 Connector 处理,进一步细化 IDA Pro 的所得程序信息,作为 CodeSurfer 的输入,得到完整、精确的中间表示。Connector 进行值集分析 VSA(Value Set Analysis)和聚集结构标识 ASI(Aggregation Structure Identification)处理,根据程序内存空间可能存放的内容和执行的操作,获得程序数据流信息。

VSA 算法根据 IDA Pro 反汇编所得内存访问信息,识别程序变量,初始 a-locs 内容。VSA 得到的间接跳转和间接函数调用信息,可以用于扩充现有的调用图和过程流程图。ASI 进一步优化 VSA 的结果,分析 a-locs 所得聚集结构类型变量(例如结构、数组、字符串等)的元素信息,并将分析结果作为 VSA 输入,开始新一轮的内存信息恢复。通过 VSA 和 ASI 迭代执行,最终得到程序每条指令处程序内存变量信息及其操作。经过优化的信息作为 CodeSurfer 的输入构建过程间控制流程图和系统依赖图,完成中间表示的构建。

3.2.1.2 模型检测

WPDS(Weighted Push Down System)[24,25] 即加权的 PDS(Push Down System)系统,是一种从 ICFG 图中间表示抽象的模型系统。PDS 模型抽象表示各个程序点状态,通过检查某个状态的可达性完成程序分析。WPDS 在 PDS 的基础上,通过扩展和合并操作用半环元素对各项迁移赋值,在原来的模型检测过程中实现数据流分析[26]。文献[27,28]使用 CodeSurfer/ x86 恢复的中间表示,构建 WPDS 系统。

这种二进制静态分析方法基于程序的"标准"编译模型假设[22]。这个"标准"模型包括,程序维护一个运行时栈,过程调用时将活动记录(AR)压栈,程序退出时将活动记录出栈,全局变量存放在固定的内存偏移位置,过程变量存放在相应活动记录的固定偏移地址,程序指令存放在固定的内存区域,程序是不可以自修改(self-modify)的,并且和程序数据分开存放。当程序不满足上述条件时,会产生错误,不能恢复程序中间表示,创建程序模型。

模型检测方法可以和程序反汇编、模式匹配、符号执行、自动机等多种方法结合使用,辅助完成模型创建和程序分析。

3.2.2 基于符号执行的缺陷检测

静态分析方法不实际执行程序,可以通过符号执行[29] 的方法模拟程序的可能执行路径。符号执行即用符号代替实际输入,解释执行程序,得到程序所有可能的执行路径,检查得到的程序路径,以发现可能存在的不安全动作,完成程序的安全性分析。这种方法在二进制程序安全分析中广泛应用,获得程序执行信息[17,20,30]。文献[20]利用 IDA Pro 所得程序信息,符号执行得到程序的执行路径。

UCSB 的 Christopher Kruegel 和 William Robertson 等人利用符号执行,将采用迷惑技术的二进制程序成功转换成符号表示[17]。他们通过检测 CFG 验证结果的正确性,这种方法的实验结果同 Objdump,Linn/Debray 的二进制代码解析方法以及 IDA Pro 的分析结果相比,具有明显优势,覆盖率平均达到了 90%并且执行速度快,处理时间和被解析二进制代码的数量成线性关系。

在文献[17]的基础上,Marco Cova 等人利用符号执行所得信息检测二进制程序,发现不安全使用 system()和 popen()函数调用引入的污点(taint)。检测过程首先标记命令行参数、环境变量以及从文件读入的数据等不安全数据源,然后通过动态连接二进制文件的过程连接表 PLT(Procedur Linkage Table)和重定位(relocation)表,追踪 system()和 popen()函数调用,当污点数据被用作这些函数的参数时,检测过程报告发现了安全威胁。污点数据从源到使用点的传递是通过符号执行技术完成的,符号执行过程根据每条指令的操作数是否为污点数据,判定其操作结果的安全性。实验数据表明,使用符号执行方法可以有效地发现程序中的污点数据[30]。

应用于二进制程序分析的符号执行过程主要面临 3 个问题:第一,路径爆炸问题,符号执行路径数随指令数成指数级增长;第二,路径中存在分支指令和循环;第三,变量别名和未知地址写操作。

随着指令数增长,路径数成指数级增长,即使一个小的程序也会有复杂的符号执行路径集合。二进制程序分析过程中,采用多种方法削减路径爆炸带来的系统开销。文献[20]检测对象为 Linux 内核可加载模块,虽然没有考虑路径爆炸问题,但性能开销仍在可接受范围内。为了解决这个问题,文献[30]使用上下文敏感的方式遍历 ICFG,得到所有与不安全调用相关的路径,而不考虑其他无关路径。根据文献[30]的实验数据,在 30 分钟内符号执行完全遍历大小 12kB 的二进制程序,遍历大小为 57.7kB 的程序的 93%,缺陷安全检测漏报率为零。

符号执行过程中会遇到分支指令,为了确定分支执行方向,在程序的执行状态中加入布尔值路径限制(path con-

strain)[31]或者路径条件(path condition)[30]。根据符号输入确定在分支点路径限制或者路径条件的值,确定分支执行方向。当程序中存在循环的时候,符号执行无法静态检测到循环结束条件。文献[31]使用基于支配树的(dominator tree) Lengauer-Tarjan 算法[32]发现控制流图中的循环,通过计算循环不动点[33,34]得到近似的循环执行结果。文献[20]使用了文献[35]支配树的算法,找到并取消程序控制结构中的后向边,从而消除了符号执行过程中可能遇到的循环指令。

机器指令将所有数据段地址空间作为相同的存储位置统一对待,写操作可能修改任意变量。二进制程序中的变量别名比高级语言程序中更难以发现,未知地址的写操作与变量别名相关,可能修改其他的存储位置。文献[36]DIVINE 利用 VSA 和 ASI 算法识别二进制程序中的变量,文献[37]在其基础上根据两个内存引用是否指向同一内存单元的记录信息,对二进制程序进行别名分析。目前二进制程序分析过程中对于变量别名都没有特别的处理。

符号执行可以用于实现路径敏感和上下文敏感的过程间分析框架,为静态获得程序执行信息提供了可能。符号执行所得路径集合可能包括实际不可行路径,会引起安全缺陷检测误报。符号执行方法在二进制程序安全缺陷检测中的应用需要进一步研究和发展。

3.2.3 基于程序注释的缺陷检测

3.2.3.1 程序注释

程序注释,即在不改变程序原来执行语义的基础上,通过代码重写修改程序,在程序执行过程中完成程序安全缺陷检测。注释代码不能干扰二进制程序执行指令的内存引用。程序注释存在 3 个关键问题:决定插入注释代码的位置,如何插入注释代码而不干扰程序的地址空间,以及注释代码和源程序执行上下文切换。

安全缺陷的可能存在位置决定插入注释代码的位置,例如文献[38]检测代码中栈相关的缓冲区溢出,所以注释代码插入到引起栈空间分配和释放操作的函数开始和结尾位置,并且这些函数含有局域变量,因为只有存在局域变量操作的时候才可能存在栈相关的缓冲区溢出。准确的定位注释代码位置需要准确的程序结构、内容表示,因此程序解析要有较高的精确度,才能够降低二进制程序分析结果的误报率。

为了尽量少地干扰程序的地址空间,应该分配独立于原程序的只读代码段用于存放注释代码。通过跳转语句,程序控制流从源程序的注释插入位置转至注释代码段,注释代码执行完毕之后跳转至注释位置继续执行接下来的代码。直接跳转指令 JMP 大小为 5 个字节,因此需要至少 5 个字节的位置用于插入转移控制流到注释代码的跳转指令。当被注释指令长度小于 5 个字节时,文献[15,38]在不修改程序的执行语义的前提下,将源程序中的指令序列放在注释代码段的前端。但是当被替代的指令序列是其他直接跳转分支地址指令时,则不能将该指令移到注释代码段,这时只能在程序中插入长度仅为 1 个字节的断点(INT3),文献[15,38]通过修改 ntdll.dll 中的 KiUserExceptionDispatcher()函数将该断点的异常处理设定为用户定义的安全检测函数。

程序注释方法的第三个困难是,注释代码执行过程中源程序执行上下文必须妥善保存以便继续源程序的执行。文献[15]采用插桩(stub)的方法解决。程序执行到注释点,由桩(stub)复责程序执行跳转至注释代码。桩完成当前系统状态的保存工作以及当注释代码执行完毕返回注释点时系统状态的恢复工作。

3.2.3.2 程序注释方法应用

程序注释方法,能够应用于多种类型的安全缺陷检测。文献[38]保护程序免于基于栈的缓冲器溢出,在 Win32/Intel 的 PE 格式程序内所有函数开始位置,通过程序注释添加返回地址保护(RAD)代码,保护栈中返回地址的完整性。程序注释方法同样适用于源码检测,文献[39]对 C 程序添加注释码,检测函数参数输入是否符合条件,发现可能出现的缓冲器溢出。文献[40]注释程序中指针别名变量的 strcpy 操作,动态检测是否存在缓冲区溢出缺陷。FOOD[12]通过对二进制程序的静态解析和程序注释,检测外部代码(Foreign code),是基于 BIRD 二进制解析框架[15]的一种检测工具,用于检测程序中非认证的控制转移。

代码注释不仅是一种缺陷发现方法,还是一种有效的缺陷修补方法。文献[38]不仅通过这种方法发现了 Windows NT4.0 中 Winhelp.exe 的一个缓冲区溢出缺陷,而且通过程序注释方法修改 Winhelp.exe 成功修补该安全漏洞。

研究者们研究和提出各种规范、框架为代码注释的广泛应用提供帮助。文献[41]提出通用汇编语言接口(Generic Assembly Language Interface)作为中间表示,为二进制程序注释提供统一规范。研究者研究出多种二进制注释框架,但大都是动态程序解析方法,例如 Valgrind[42],PIN[43],Dynmo-RIO[44]等。另外一种静态动态结合的二进制代码解析工具 BIRD[15]保证解析精确度达到 100%,可以作为辅助程序注释的框架。这种方法保守地使用递归遍历从二进制程序的入口点静态解析所有直接可达的"正确指令",对于其他的"可能指令",使用积分规则判断其正确性。其规则的制定结合实验数据,根据不同应用分配不同的积分,例如函数 prolog 模式匹配能够明显保证正确解析结果,因此匹配函数 prolog 模式的可能指令具有最高积分值(8)。这样能够在增大解析覆盖率的同时保证正确性。使用源程序作为参照评测 BIRD 静态反汇编算法的有效性。应用程序源码通过 VC6.0 编译选项,得到中间汇编表示和包含详细符号信息的程序数据库 PDB(program database)文件。通过比较发现,相应二进制程序通过 BIRD[15]静态代码解析的指令正确性达到 100%,但是,覆盖率仅为 69%~96%,需要通过进一步改进其中的动态分析方法增大覆盖率。

程序静态分析过程中插入的注释代码,自动执行完成程序安全性分析工作。程序执行过程有助于提高静态分析的精确性。因此,程序注释分析方法综合利用静态分析和动态分析方法,能够提高二进制程序安全缺陷检测的精确度和覆盖率。

结束语 目前,市场上的二进制程序安全分析工具远不能满足程序安全性需求。由于静态程序分析具有覆盖率高、没有程序执行开销和可以提前验证程序安全性等优势,二进制程序静态分析是需要进一步研究的重要研究领域。本文对二进制程序安全缺陷静态分析方法进行了综述和分析。现有分析方法可以分为基于程序结构和程序语义两类。基于程序结构的缺陷检测完全依赖于程序头信息、表信息,覆盖率和精确度低,分析结果不能作为程序安全性分析的准确度量。基

于程序语义二进制程序分析方法结合静态分析和模型检测、符号执行以及程序注释等方法，根据程序信息构建合适的中间表示，在中间表示中发现程序的安全缺陷，其覆盖率和精确度都较高，这是二进制程序安全分析的主要研究方向。

基于程序语义的二进制程序安全分析主要有三种方法：模型检测、符号执行和程序注释。目前，大部分模型检测方法仅能用于简单的二进制程序检测，例如 Linux 内核可加载模块的 rootkit[20]检测。WiSA[9]致力于开发二进制程序静态分析平台，能够分析复杂软件系统，但是它们恢复程序中间表示过程建立在"标准模型"假设的基础上，不具备通用性。因此，需要进一步研究，得到完整的、通用的中间表示，模型检测方法才能更好地应用于二进制程序分析。符号执行方法在程序静态分析中广泛应用，但是符号执行的"路径爆炸"问题一直都没有得到较好的解决，限制了静态分析的程序规模，另外，符号执行如何解决间接跳转和条件分支等问题也需要进一步研究。程序注释是程序静态和动态分析结合的一种方法，目前已经有一些程序注释平台提供统一的程序注释框架。

基于程序语义的二进制程序安全分析方法必须首先恢复程序信息。二进制程序的中间表示构建是整个程序分析过程中的关键。寻找一种提供丰富程序信息的通用的中间表示，研究更高精确度和覆盖率的分析算法是下一步研究的主要问题。

参 考 文 献

[1] Balakrishnan G. WYSINWYX: What You See is Not What You Execute[D]. State of Winsconsin: University of Wisconsin, 2007

[2] HBGary, HBGary Inc. [OL]. http://www.hbgary.com/. accessed on Nov. 2007

[3] @stake, Symantec to Acquire @stake[OL]. http://www.symantec.com/press/2004/n040916b.html accessed on Nov. 2007

[4] Grehan R. Product Review: BugScan[OL]. http://www.itworldcanada.com/product/2/47224.html. accessed on Nov. 2007

[5] Dragan R V. SmartRisk Analyzer 1.0-Reviews by PCMagzine [OL]. http://www.pcmag.com/article2/0,2817,1625399,00.aspaccessed on Nov. 2007

[6] Durden T. Automated Vulnerability Auditing in Machine Code [J]. Phrack, 2006

[7] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C]// Network and Distributed System Security Symposium(NDSS). 2005

[8] Bergeron J, et al. Static detection of malicious code in executable programs[J]. Int. J. of Req. Eng., 2001

[9] Wisconsin Safety Analyzer [OL]. http://www.cs.wisc.edu/wisa/about.html

[10] Tevis J E J. Automatic Detection of Software Security Vulnerabilities in Executable Program Files. Auburn University

[11] Tevis J E J, Hamilton J A Jr. Static analysis of anomalies and security vulnerabilities in executable files[C]// Proceedings of the 44th annual southeast regional conference. 2006: 560-565

[12] Nanda S, Chiueh T C. Foreign code detection for Windows/X86 binaries[R]. ECSL TR-190. Computer Science Department, Stony Brook University, 2005

[13] IDA[OL]. http://www.datarescue.com/

[14] Jha M C S. Static Analysis of Executables to Detect Malicious Patterns

[15] Nanda S, et al. BIRD: Binary Interpretation using Runtime Disassembly[C]// Proceedings of the International Symposium on Code Generation and Optimization. 2006: 358-370

[16] Schwarz B, Debray S, Andrews G. Disassembly of executable code revisited[C]// Reverse Engineering, 2002. Proceedings. Ninth Working Conference. 2002: 45-54

[17] Kruegel C, et al. Static Disassembly of Obfuscated Binaries[C]// USENIX. 2004

[18] Corbett J C, et al. Bandera: extracting finite-state models from Java source code[C]// Software Engineering, 2000. Proceedings of the 2000 International Confe-rence. 2000: 439-448

[19] Courses E, Surveys T. A Portable Compiler - Integrated Approach to Permanent Checking[C]// Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on. 2006: 103-112

[20] Kruegel C, Robertson W, Vigna G. Detecting kernel-level rootkits through binary analysis[C]// Computer Security Applications Conference, 2004. 20th Annual. 2004: 91-100

[21] Reps T, Balakrishnan G, Lim J. Intermediate-representation recovery from low-level code[C]// Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation. 2006: 100-111

[22] Reps T, et al. A next-generation platform for analyzing executables[C]// Programming Languages and Systems, Proceedings. 2005, 3780: 212-229

[23] GrammaTech[OL]. http://www.grammatech.com/products/codesurfer/

[24] Reps T, Lal A, Kidd N. Program Analysis Using Weighted Pushdown Systems[C]// Lecture Notes in Computer Science. 2007, 4855: 23

[25] Lal A, Reps T. Improving pushdown system model checking. [C]// Computer Aided Verification, Proceedings. 2006, 4144: 343-357

[26] Lal A, Reps T, Balakrishnan G. Extended weighted pushdown systems[J]. Computer Aided Verif, 2005

[27] Reps T, et al. Weighted pushdown systems and their application to interprocedural dataflow analysis[J]. Science of Computer Programming, 2005, 58(1/2): 206-263

[28] Balakrishnan G, et al. Model checking x 8 6 executables with CodeSurfer/x86 and WPDS++[C]// Computer Aided Verification, Proceedings. 2005, 3576: 158-163

[29] King J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394

[30] Cova M, et al. Static Detection of Vulnerabilities in x86 Executables[C]// Proceedings of the Annual Computer Security Applications Conference(ACSAC). Miami, FL, December 2006

[31] Kruegel C, et al. Automating mimicry attacks using static binary analysis[C]// Proceedings of the 14th USENIX Security Symposium. 2005

[32] Lengauer T, Tarjan R E. A fast algorithm for finding dominators in a flowgraph[J]. ACM Transactions on Programming Languages and Systems(TOPLAS), 1979, 1(1): 121-141

[33] Cousot P, Cousot R. Abstract interpretation: a unified lattice

model for static analysis of programs by construction or approximation of fixpoints[C]// Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977:238-252

[34] Nielson F, Nielson H R, Hankin C. Principles of Program Analysis[M]. Springer, 1999

[35] Aho A V, Sethi R, Ullman J D. Compilers: principles, techniques, and tools[M]. Inc. Boston, MA, USA: Addison-Wesley Longman Publishing Co., 1986

[36] Balakrishnan G, Reps T. DIVINE: DIscovering Variables IN Executables. VMCAI, 2007: 1-28

[37] Brumley D, Newsome J. Alias analysis for assembly[R]. CMU-CS-06-180. Carnegie Mellon University School of Computer Scie-nce, 2006

[38] Prasad M, Chiueh T. A binary rewriting defense against stack based buffer overflow attacks[C]// Proceedings of the USENIX Annual Technical Conference. 2003: 211-224

[39] Haugh E, Bishop M. Testing C Programs for Buffer Overflow Vulnerabilities[C]// Proceedings of the Network and Distributed System Security Symposium. 2003

[40] Aggarwal A, Jalote P. Integrating Static and Dynamic Analysis for Detecting Vulnerabilities[C]// Proceedings of the 30th Annual International Computer Software and Applications Conference(COMPSAC'06). Volume 01, 2006: 343-350

[41] Harris L C, Miller B P. Practical analysis of stripped binary code [J]. ACM SIGARCH Computer Architecture News, 2005, 33 (5): 63-68

[42] Nethercote N. Dynamic Binary Analysis and Instrumentation [D]. University of Cambridge, UK, 2004

[43] Luk C K, et al. Pin: building customized program analysis tools with dynamic instrumentation [C] // Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. 2005: 190-200

[44] Dynamorio[OL]. http://www.cag.lcs.mit.edu/dynamorio

(上接第 7 页)

[19] Zapata M G. Secure ad hoc on-demand distance vector routing [J]. Mobile Computing and Communications Review, 2006, 6 (3): 106-107

[20] Eichler S, Roman C. Challenges of secure Routing in MANETs: A Simulative Approach using AODV-SEC[C]// IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS2006). 2006: 481-484

[21] Liu Jun, Li Zhe, Lin Dan, et al, A Security Enhanced AODV Routing Protocol Based On the Credence Mechanism[C]// International Conference on Wireless Communications, Networking and Mobile Computing 2005. 2005, 2: 719-722

[22] Li Leiyuan, Chigan C. Token Routing: A Power Efficient Method for Securing AODV Routing Protocol[C]// Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, (ICNSC '06). 2006: 29-34

[23] Rifa-Pous H, Herrera-Joancomarti J. Secure Dynamic MANET On-demand(SEDYMO) Routing Protocol[C]// Fifth Annual Conference on Communication Networks and Services Research, (CNSR '07). 2007: 372-380

[24] Wang M, Lamont L, Mason P, et al. An Effective Intrusion Detection Approach for OLSR MANET Protocol[C]// 1st IEEE ICNP Workshop on Secure Network Protocols, (NPSec). 2005: 55-60

[25] Kim PJihye, Tsudik PGene. SRDP, securing route discovery in DSR[C]// Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services2005. 2005: 247-260

[26] Van Der Merwe J, Dawoud D, Mcdonald S. A survey on peer-to-peer key management for mobile ad hoc networks [J]. ACM Computing Surveys, 2007, 39(1): 1-45

[27] Zhou R, Haas Z J. Securing ad hoc networks[J]. IEEE networks (Special Issue on Network Security), 1999, 13(6): 24-30

[28] Yi S, Kravets R. MOCA: Mobile certificate authority for wireless ad hoc networks[C]// Proceedings of the 2nd Annual PKI Research Workshop(PKI 2003)

[29] Kong J, Zerfos P, Luo H, et al. Providing robust and ubiquitous security support for mobile ad-hoc networks[C]// Proceedings of the Ninth International Conference on Network Protocols 2001(ICNP'01)

[30] Luo H, Zerfos P, Kong J, et al. Self-securing ad hoc wireless networks[C]// Proceedings of the Seventh International Symposium on Computers and Communications 2002(ISCC'02)

[31] Joye M, Yen S M. ID-based secret-key cryptography[J]. ACM Operat. Syst. Rev., 1998, 32(4): 33-39

[32] Boneh D, Franklin M. Identity-based encryption from weil pairing[C]// Proceedings of the Conference on Advances in Cryptology 2001(CRYPTO'01)

[33] Cha J C, Cheon J H. An identity-based signature from gap diffie-hellman groups[C]// Proceedings of the Conference on Public Key Cryptography 2003(PKI'03)

[34] Capkun S, Buttyan L, Hubaux J. Self-organized public-key management for mobile ad hoc networks[J]. IEEE Transactions on Moblie Computing, 2003, 2(1): 52-64

[35] Ngai E C H, Lyu M R, Chin R T. An authentication service against dishonest users in mobile ad hoc networks[C]// Proceedings of the IEEE Aerospace Conference. 2004

[36] Eschenauer L, Gligor V D. A key-management scheme for distributed sensor networks[C]// Proceedings of the 9th ACM Conference on Computer and Communication Security 2002 (CCS'02)

[37] Capkun S, Buttyan L, Hubaux J. Mobility helps security in ad hoc networks[C]// Proceedings of MobiHoc 2003

[38] Capkun S, Hubaux J, Buttyan L. Mobility helps peer-to-peer security[J]. IEEE Transactions on Moblie Computing, 2006, 5 (1): 43-51

[39] Yi S, Kravets R. Composite key management for ad hoc networks[C]// Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Network and Services(MobiQuitous'04)

[40] Shin K, Kim Yoonho, Kim Yanggon. An Effective Authentication Scheme in Mobile Ad Hoc Network[C]// Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing(SNPD'06). 2006: 249-252

[41] Nikodem J, Nikodem M. Secure Communication Trees in Ad Hoc Networks[C]// Proceedings of the 14th Annual IEEE International Conference and Workshops on Engineering of Computer-Based Systems 2007(ECBS'07)

[42] Sen J, Chowdhury P R, Sengupta I. A Distributed Trust Establishment Scheme for Mobile Ad Hoc Networks[C]// International Conference on Computing: Theory and Applications, (ICCTA '07). 2007: 51-58

[43] Tanabe M, Aida M. Preventing Resource Exhaustion Attacks in Ad Hoc Networks[C]// Eighth International Symposium on Autonomous Decentralized Systems(ISADS'07). 2007: 543-548