一种基于模型检测的二进制程序脆弱性分析框架

王春雷[1,2] 刘 强[2] 赵 刚[1,2] 戴一奇[2]
(清华大学计算机科学与技术系 北京 100084)[1] (北京系统工程研究所网络技术研究室 北京 100101)[2]

摘 要 针对二进制程序脆弱性分析的实际需求,提出了一种基于模型检测的二进制程序脆弱性分析框架。首先定义了二进制程序的抽象模型,描述了基于有限状态自动机的软件脆弱性形式化表示和基于事件系统的软件安全属性表示方法。在此基础上,提出了基于模型检测的脆弱性分析过程和算法。根据该分析框架,设计并实现了二进制程序脆弱性分析工具原型。通过脆弱性分析实验,详细说明了该框架的工作原理,验证了该分析方法的有效性。

关键词 模型检测,二进制程序,脆弱性分析,形式化方法
中图法分类号 TP312 **文献标识码** A

Vulnerability Analysis Framework for Binaries Based on Model Checking

WANG Chun-lei[1,2] LIU Qiang[2] ZHAO Gang[1,2] DAI Yi-qi[2]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[1]
(Laboratory of Network Technology, Beijing Institute of System Engineering, Beijing 100101, China)[2]

Abstract In order to analyze vulnerabilities in executable programs, a vulnerability analysis framework for binaries based upon model checking was proposed. Firstly, the abstract model of binary was defined, and the formal models of vulnerabilities based upon finite state automaton and the representations of software security attributes based upon event system were described. Then, the model checking based vulnerability analysis process and algorithm were proposed with respect to the abstract models of binaries and the security attributes to be checked. After that, the prototype of vulnerability analysis tool was designed and implemented based upon the framework. The illustrative sample program was analyzed to show in detail the principles of the framework, and the experimental results show the effectiveness of the analysis method.

Keywords Model checking, Vulnerability analysis, Formal method

1 引言

确保信息系统安全性的重要前提之一是准确理解系统中存在的脆弱性,这些脆弱性可以被利用从而破坏系统的安全性。当前,安全性分析方法更多地是使用直接的手工代码审计、日志文件分析以及代码分析工具等。由于信息系统的复杂性和脆弱性的多样性,使得这些方法具有很大的缺陷,并且具有很高的误报率和漏报率。语义分析方法(例如静态分析可执行程序)提供了一种克服这些问题的可能途径。因此,如何通过形式化方法分析软件系统,特别是二进制程序的脆弱性是需要研究的重要问题。

在攻击行为建模研究领域,目前有多个研究小组开展相关研究并且取得了阶段性进展。Singh 和 Lakhotia 描述了一个形式系统,该系统使用模型检测器 SPIN 检查可疑程序的控制流图属性,该属性通过线性时态逻辑(LTL)公式指定潜在的恶意行为[1]。Sheyner 和 Haines 等研究并提出了一个基于有限状态自动机(FSM)的攻击图自动构造技术[2]。Michael 和 Ghosh 使用系统调用轨迹构造了一个有限状态自动机模型,通过使用正常轨迹训练该模型,使其具有识别程序异常行为的能力,从而达到检测入侵的目的,而形式化验证系统的安全性是通过符号模型检测器实现的[3]。

模型检测方法已经被成功地用于硬件和软件的验证[4,5]。将模型检测方法运用于系统脆弱性分析研究,已经逐渐得到信息安全研究领域的重视。其中,Ramakrishnan 和 Sekar 描述了一种基于模型检测的系统配置脆弱性分析方法,通过分析 UNIX 系统中的配置脆弱性,验证了该方法的有效性[6]。Chen 和 Wagner 提出了一种基于模型检测的用于发现软件缺陷的形式化方法,该方法将被验证程序建模为一个下推自动机,将安全属性表示为有限状态自动机,并且使用模型检测方法识别任何违反期望安全目标的状态是否在程序中可达[7]。R. Ritchey 和 P. Ammann 针对网络系统中由于各种主机配置而导致的脆弱性,提出了一种基于模型检测的脆弱性分析方法[8]。

分析当前的研究现状,一方面,从安全属性角度建模和形

到稿日期:2009-05-20 返修日期:2009-07-23 本文受国防预先研究项目(513150601)资助。
王春雷(1977—),男,博士生,主要研究方向为软件安全等,E-mail:wcl08@mails.tsinghua.edu.cn;刘 强(1979—),男,硕士,主要研究方向为软件安全等;赵 刚(1969—),男,研究员,主要研究方向为网络安全;戴一奇(1946—),男,教授,博士生导师,主要研究方向为密码学和网络信息安全。

式化分析软件系统，特别是针对二进制程序的脆弱性，这方面的研究工作相对较少，尚未取得明显进展。另一方面，模型检测方法和工具在脆弱性分析方面的作用已经得到初步验证。因此，二进制程序脆弱性分析采用抽象程序建模和模型检测相结合的方式可以充分利用两种方法的优点。本文提出了一种新的基于模型检测的二进制程序脆弱性分析方法。该方法通过检查程序规范和实现之间安全断言的违反情况，从而精确地检测和分析应用程序中存在的脆弱性。

本文第2节介绍二进制程序脆弱性的形式化建模方法，主要包括二进制程序抽象模型和基于有限状态自动机模型的脆弱性表示方法。第3节提出二进制程序脆弱性分析框架，主要包括软件安全属性表示以及基于模型检测的脆弱性分析过程和算法。第4节介绍原型实现和脆弱性分析实验，验证本框架的有效性。最后，总结全文并指出下一步的工作。

2 二进制程序脆弱性的形式化建模

2.1 二进制程序抽象模型

通过模型检测方法分析软件系统的脆弱性，首先需要提供关于被分析系统的形式化模型。在本节中，根据二进制程序的特点和脆弱性分析的需要，定义二进制程序的抽象模型，描述从指令序列中构建二进制程序抽象模型的方法。该方法从程序的入口点集合出发，通过自底向上将指令序列聚集成更大的单元，如基本块和例程，重构二进制程序的执行模型。然而，由于现代计算机的体系结构非常复杂，从二进制代码中精确地重构出程序的执行模型是非常困难的，例如，内存重定向、过程变量以及机器指令的模糊使用等问题。因此，我们根据脆弱性分析的需要，重点关注影响执行流程和数据处理的程序状态和状态转换，提出二进制程序抽象模型及其构建方法。

设 S 为一个集合，$|S|$ 表示 S 的基数，$p(S)$ 表示 S 的幂集。抽象机器描述可以表示为 $M=(I,c,j)$，其中 I 为指令集，$c:I\to p(Addrs)$ 为指令到它们的调用目标地址的映射，$j:I\to p(Addrs)$ 为指令到它们的目标地址和直接后继指令地址（如果可达）的映射。假设 c 和 j 可以从 i 独立地计算得到，$i\in I$。也就是说，一条指令包含关于目标地址的所有信息，而不需要进一步地查找内存。

定义 1 机器 M 的二进制抽象程序模型为：

$$P(M)=(A,e,s) \tag{1}$$

其中，$A\subseteq Addrs$，$|A|$ 为包含指令地址的有限集合，$e\in A$ 为程序的入口地址，$S:A\to I$ 为从地址到指令的映射，即内存的内容。

这里，假设不存在跳转或调用离开二进制可执行程序的情况，因此将 c 和 j 限制到 $I\to p(A)$ 中。而且，c 和 j 应该是可计算确定的分支目标。

为了构建二进制程序的抽象模型，必须知道下列信息：

(1)例程入口地址的集合，$R\subseteq A$；

(2)对于每个 $r\in R$，从 r 开始的例程所包含的所有指令地址的集合，$b(r)\subseteq A$。

设 f 和 g 为函数，$f\cdot g$ 表示函数的组合，即 $(f\cdot g)(x)=f(g(x))$，则通过 $j\cdot s$ 和 $b(r)$，可以完整地描述每个例程的抽象模型。通常，一个二进制程序是由例程的集合和对于例程的调用组成的。通过 c 和 R，可以描述程序的调用情况，因为当运用至每个 $b\in b(r)$，$r\in R$ 时，$c\cdot s$ 描述该二进制程序的所有调用边。

序列 (R_n,b_n) 用于描述已定义的 R 和 b。二进制程序的第一个例程是从入口点 $R_0=\{e\}$ 开始的。设

$$b_0:R_0\to A$$

$$r\mapsto\{r\}$$

对于 R_{k+1}，$c\cdot s$ 从 $b_k(R_k)$ 发现例程入口：

$$R_{k+1}:=R_k\cup\bigcup_{r\in R_k}\bigcup_{b\in b_k(r)}c(s(b)) \tag{2}$$

对于已知的例程，通过 $j\cdot s$ 可以发现调用边。而对于新的例程，使用如下的例程入口地址：

$$b_{k+1}:R_{k+1}\to A$$

$$r\mapsto\begin{cases}b_k(r)\cup\bigcup_{b\in b_k(r)}j(s(b)), & \text{if } r\in R_k\\ \{r\}, & \text{else}\end{cases} \tag{3}$$

通过迭代地执行上述过程，可以自底向上逐步构建被分析的二进制程序的抽象模型。

2.2 基于有限状态机的脆弱性表示

软件脆弱性分析通常包括理解程序结构、跟踪程序行为和识别其中的安全缺陷。而软件脆弱性建模则是该分析过程的基础。例如，Sheyner 的攻击图构造器[2]，通过建模脆弱性和攻击特性提供一定程度的形式化。本文提出一个有限状态自动机（FSM）模型，用于形式化建模和分析软件系统的脆弱性。

通过分析各种脆弱性实例，可以发现通常攻击必须经过多个基本活动，而在其中任何一个位置都可以有效地阻止攻击，因为特定脆弱性的利用过程通常涉及到多个对象上的若干不安全的操作。根据安全性需求，结合不同类型脆弱性的特征和检测方法，允许我们指定确保安全性所需要的断言。

根据上述观察和特性分析，我们提出一个基于有限状态自动机（FSM）的脆弱性建模方法。方法通过将脆弱性利用过程分解为多个操作，每个操作包括一个或多个基本活动，从而能够表示完整的脆弱性利用过程。由于每个基本活动是简单的，因此开发安全断言和相应的基本型有限状态自动机（eFSM）来表示基本活动是可行的。然后，通过方便地组合这些 eFSM 来构造 FSM 模型，用于表示不安全的操作和可能的利用过程。

该 FSM 的目标是证明已实现的操作以及操作中的每个基本活动是否满足已推导的安全断言。这里，采取3个步骤：

(1)将每个基本活动表示为一个 eFSM，它表达用于接受某输入对象的断言；

(2)将一个针对某对象的操作建模为一个 eFSM 序列；

(3)通过将不同的操作进行层叠处理，实现软件系统脆弱性的建模。

因此，eFSM 是我们所提出的程序模型中的基本构造块，定义如下。

定义 2 一个程序活动抽象为一个基本 FSM（eFSM）：

$$M=(Q,\Sigma,\delta,q_0,F) \tag{4}$$

其中，Q 是活动内部状态的初始集合，Σ 是表示程序基本块符号的有限集合，$\delta:Q\times\Sigma\to Q$ 是一个转换函数，$q_0\in Q$ 是用于检查的状态（即程序活动的初始状态），而 $F\subseteq Q$ 是程序活动的结束状态集合。

首先，针对二进制程序构造抽象模型和有限状态自动机；

然后，根据安全规范设置安全断言；最后，针对模型检查安全断言。如果在实现中出现一个违反已推导断言的情况，那么将定位相关的脆弱性。eFSM 包括 4 个转换和 3 个状态，如表 1 所列。

表 1　eFSM 中包括的转换和状态

类型	名称	描述
转换	S_ACPT	关于接受对象(如，一个用户或一个请求)的规范断言
	S_REJT	关于拒绝对象的规范断言
	I_REJT	在实现中关于拒绝的情况，而根据规范该拒绝是应该发生的，这是期望的或正确的行为
	I_ACPT	在实际实现中一个对象被接受的情况，而根据规范该对象应该被拒绝，这表示一个脆弱性的隐藏路径
状态	检查状态	根据规范检查一个对象
	拒绝状态	根据规范，转换到拒绝状态指示该对象是不安全的
	接受状态	转换到接受状态指示该对象是安全的

由于每个基本活动是简单的，因此通过使用二进制程序抽象模型构造相应的 eFSM 和安全断言是可行的。然后，可以方便地组合 eFSM 以描述 FSM，从而建模脆弱性操作和可能的利用。eFSM 的典型结构如图 1 所示。

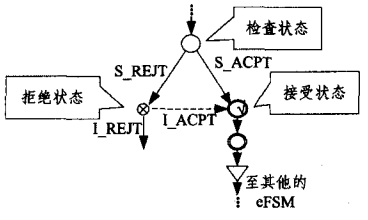

图 1　基本 FSM (eFSM)

该 FSM 模型引入一个传播通道的概念(表示为 eFSM 之间的▽标志)，用于描述在两个操作之间的脆弱性利用的因果关系。通过迭代执行该过程，最终形成完整的程序模型。

3　二进制程序脆弱性分析框架

在二进制程序脆弱性形式化建模的基础上，以模型检测技术为分析手段，构建二进制程序脆弱性分析框架。该框架提供安全属性的表示方法，并且基于模型检测工具提供二进制程序脆弱性分析的一般性过程。

3.1　安全属性表示

程序脆弱性是指违反某种安全属性的行为或状态。通常，从形式化的角度，对于验证程序 P 的属性，即模型 M_P 是否正确地反映程序的执行流程，必须要求模型 M 包括 P 的所有可能的执行轨迹。从本质上说，这是从程序 P 构造模型 M 进行安全性分析的正确性准则。

这里，我们主要关注程序模型的安全属性。安全属性是执行程序上的断言，而所有的安全属性都可以被看作是安全策略的实例化[9]。针对特定的程序，如果一个安全属性对于每个可能的执行轨迹都保持为真，那么该程序满足该安全属性。根据文献[9,10]，可以将程序抽象为一个事件系统，事件系统是通过事件与其环境交互的。这些事件对应于程序的基本活动，即 2.2 节定义的基本 FSM(eFSM)，而一个事件序列对应于系统中的一个可能的执行序列，称为轨迹(trace)。

定义 3　一个事件系统是一个四元组：

$$S=\langle E,O,I,T\rangle \tag{5}$$

其中，E 是事件集合；I 是输入事件，$I\subseteq E$；O 是输出事件，$O\subseteq E$ 并且 $I\cap O=\phi$；T 是轨迹集合。

对于给定的系统 S 和轨迹 τ，与 τ 具有相同低级事件的轨迹集合为：

$$LTS(\tau,S)=\{s|\tau|L=s|L\wedge s\in traces(S)\} \tag{6}$$

其中，$\tau|L$，$s|L$ 表示从 τ，S 中删除所有不在 L 中的事件构成的轨迹，$traces(S)$ 表示 S 的轨迹集合。

从事件系统的角度看，安全属性的目标是阻止低层用户能够推导关于高层用户的事件。而低层用户不应该具备哪些推导，取决于该安全属性试图实施的信息流策略。也就是说，安全属性确保特定轨迹是 $LTS(\tau,S)$ 的元素。针对一个安全属性，如果一个系统所有要求的轨迹都出现在具有相同低级事件的轨迹集合 $LTS(\tau,S)$ 中，则系统满足该安全属性。

通过基于事件系统的安全属性表示形式，可以形式化定义各种常见的安全属性，如不干扰性、可隔离性等信息流安全属性等。这些安全属性可以支持安全断言的构建，从而为基于模型检测的脆弱性分析提供约束条件。

3.2　模型检测过程

通常，模型检测规范包括两个部分：一部分是模型，即根据变量、变量初始值以及变量发生变化的条件而定义的状态机；另一部分是关于状态和执行路径的时态逻辑约束。从概念上讲，模型检测器访问所有可达的状态并且验证时态逻辑属性在每条执行路径上的满足情况。也就是说，模型检测器确定有限状态机是否为一个符合时态逻辑公式的模型。如果一个属性没有被满足，则模型检测器试图以轨迹或状态序列的形式生成相应的反例。

基于模型检测的形式化方法已经在学术研究文献中得到相当多的关注，而目前可用的工具如 SPIN[11] 和 SMV[12]，已经能够处理与现实问题相关联的状态空间。本文使用 SPIN 模型检测器作为二进制程序脆弱性分析的基础。SPIN 工具是 20 世纪 80 年代开始由贝尔实验室计算机科学研究中心 Unix 小组开发的。模型检测尽管最初是作为一种验证硬件设计的方法，但是也可以被应用到大型软件系统规范的自动化验证，例如 LaRS[13]。

模型检测器在软件开发尤其是软件安全性分析方面，已经成为一个有吸引力的形式化分析检测工具。因此，将二进制程序抽象为有限状态自动机描述(见第 2 节)，再通过时态逻辑形式编写断言，以表达特定的安全属性。然后，使用模型检测器验证该断言在模型中保持为真，或者生成针对该软件的攻击场景，以显示攻击者是如何突破系统的。模型检测器所支持的时态逻辑提供一个用于指定安全性需求的丰富的语言，即安全策略需求，例如访问控制或信息流安全，能够相对直接地转换为时态逻辑。使用模型检测进行脆弱性分析的一般性过程如图 2 所示。

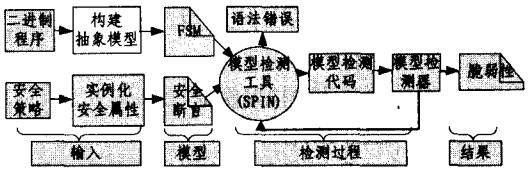

图 2　基于模型检测的二进制程序脆弱性分析过程

3.3　基于模型检测的脆弱性分析算法

该框架中所采用的脆弱性分析算法是对基于时态逻辑抽象的模型检测算法的拓展。为了分析二进制程序中的脆弱性，首先利用 2.2 节中描述的二进制程序抽象建模的方法获

得以有限状态自动机形式表示的被分析二进制程序的抽象模型 M。然后，根据被分析程序的安全策略和脆弱性分析的需求，确定该程序的安全属性集合 *attr_set*，将模型 M 和集合 *attr_set* 作为脆弱性分析算法的输入。

以抽象模型 M 为对象，该算法针对安全属性集合 *attr_set* 中的每个安全属性 *attr*，运用模型检测方法逐个进行分析。该算法的目的是通过进一步的分析，从 *attr_set* 中识别潜在不安全的 *attr*，将其作为脆弱性分析的依据。该分析算法的主要流程如图 3 所示。

```
foreach attr in attr_set do {
  // ModelChecking():模型检测方法
  // IN 参数:M 为抽象模型,attr 为待检测的安全属性
  // OUT 参数:如果 ModelChecking()输出为 false 时,trace 为相应的违反安全属性 attr 的轨迹
  // 返回值:布尔值,true 表示该属性通过模型检测,false 表示模型 M 中存在违反安全属性的轨迹
  boolean result = ModelChecking(M,attr,trace);
    if (result = false)
    vul_list.add(attr,trace);
}
foreach vul in vul_list do {
  // 打印脆弱性执行轨迹
  print_vul_trace(vul);
}
```

图 3　基于模型检测的二进制程序脆弱性分析算法

其中，*ModelChecking*(M, *attr*, *trace*)的目标是分析 *attr* 在二进制程序抽象模型 M 中的可达性。如果经过模型分析，发现其中存在违反该安全属性的轨迹，则该 M 就会被添加到脆弱性结果集合 *vul_list* 中。同时加入到 *vul_list* 中的还有相应的错误执行轨迹，即从可达树的根结点到违反 *attr* 属性结点的路径，用于进一步分析程序脆弱性的来源和触发机制，并且可以作为对该二进制程序进行脆弱性动态分析和测试的依据。

对程序进行合理的抽象是 *ModelChecking*()方法实现的基础。基于 2.2 节中描述的二进制程序的抽象模型，构建可达树的过程采取的是对基于程序的过程调用和过程内指令流构成的状态转换图的深度优先遍历过程。

在构建可达树的过程中，每当到达一个违反属性 *attr* 的结点位置 *location*，*ModelChecking*()方法会暂停可达树的构建，转而分析从可达树的根到 *location* 的执行轨迹 *trace*。通过对 *trace* 上的安全断言状态条件的分析结果，判定该轨迹是否确实存在违反指定安全属性 *attr* 的情况。以 3.1 节中描述的安全属性表示方法为依据，通过线性时态逻辑(LTL)表示属性 *attr*，而将可达树的轨迹表示为操作序列的形式。轨迹上的操作序列指定了该二进制程序到达脆弱性的状态转换过程。

4　框架实现与实例分析

4.1　框架实现

在二进制程序脆弱性分析框架的基础上，初步实现了脆弱性分析工具原型(Model-based Binary Vulnerability Analyzer，MBVA)。MBVA 使用了模型检测器 SPIN 作为脆弱性的分析引擎。MBVA 前端主要包括二进制代码的反汇编、控制流、数据流分析、基于函数签名的缺陷函数识别等模块，以及建立在缺陷函数识别基础上的代码抽取、切片、建模和脆弱性规则生成功能。MBVA 前端的模块结构如图 4 所示。

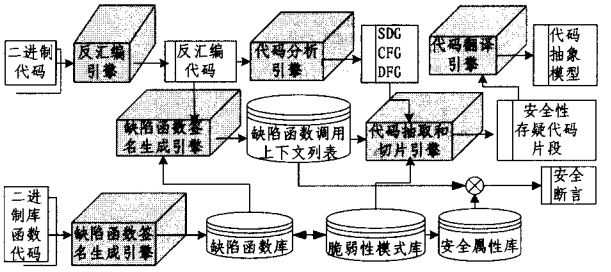

图 4　MBVA 前端模块结构示意图

其中，脆弱性规则是脆弱性模式的抽象，它对应于进行模型检查的安全属性。通过建模生成的二进制程序有限自动机描述和安全属性作为脆弱性分析引擎的输入。如果模型检测过程给出了违反安全属性的反例，则 MBVA 给出这一反例的路径，并从相应的脆弱性模式角度进行分析。

4.2　实例分析

通过 MBVA 工具原型，我们分析了已经公开发布的多种类型的脆弱性，初步验证了该分析框架的有效性，这些脆弱性类型包括堆溢出、栈缓冲区溢出、整数溢出、文件竞争条件以及格式化字符串脆弱性等，其中既包括可以被远程利用的脆弱性，也包括可以被本地用户利用的脆弱性。

在本文中，选择 Microsoft MFC 库 CFileFind::FindFile() 堆溢出脆弱性[14]详细描述基于模型检测的二进制程序脆弱性分析过程。在该实例中，程序抽象模型和相关的安全属性断言是通过 MBVA 工具原型分析该脆弱性所对应的.dll文件和脆弱性模式而获得的。

在 Microsoft Windows 操作系统中，MFC42 和 MFC71 库的 CFileFind 类在处理 FindFile()函数参数时存在堆溢出漏洞，本地攻击者可能利用此漏洞提升自己的权限。CFileFind::FindFile()函数的汇编代码如图 5 所示。

```
MFC[42|71].dll@CFileFind::FindFile(char const *,unsigned long)
.text:73D6CD3F  mov     edi, edi
.text:73D6CD41  push    ebp
.text:73D6CD42  push    esi                     ; unsigned int
.text:73D6CD43  push    edi                     ; unsigned __int8 *
.text:73D6CD44  mov     esi, ecx
.text:73D6CD46  call    CFileFind::Close(void)
.text:73D6CD4B  push    140h                    ; int  << 320 bytes
.text:73D6CD50  call    @operator new(uint)     << buffer Allocate ①
.text:73D6CD55  mov     ebp, [esp+14h]
.text:73D6CD59  and     dword ptr[esi+10h], 0
.text:73D6CD5D  test    ebp, ebp
.text:73D6CD5F  pop     ecx
.text:73D6CD60  mov     [esi+8], eax
.text:73D6CD63  jnz     short loc_73D6CD6A
.text:73D6CD65  mov     ebp, offset a__1        ; "*.*" << si arg_0 == NULL
.text:73D6CD6A  loc_73D6CD6A ; CODE XREF: CFileFind::FindFile(char const*,ulong)+24j
.text:73D6CD6A  push    ebp                     ; lpString2
.text:73D6CD6B  add     eax, 2Ch
.text:73D6CD6E  push    eax                     ; lpString1
.text:73D6CD6F  call    ds:__imp__lstrcpyA@8    ; lstrcpyA(x,x) << ②
.text:73D6CD75  push    dword ptr[esi+8]        ; lpFindFileData
.text:73D6CD78  push    ebp                     ; lpFileName
.text:73D6CD79  call    ds:__imp__FindFirstFileA@8 ; FindFirstFileA(x,x)
[...]
```

图 5　CFileFind::FindFile()函数的汇编代码

如图 5 所示，FindFile()函数为①处的缓冲区分配内存，然后未经检查便储存②处函数的第一个参数的内容。如果用户提交了超长参数就可以触发堆溢出，导致执行任意指令。

根据上面的汇编代码，首先使用 MBVA 工具抽取程序抽象模型，通过使用 eFSM 语义描述构造 CFileFind::FindFile()堆溢出脆弱性的触发过程，如图 6 所示。针对每个转换关

联一个对应的 Condition/Action 标签，其中 Condition 表示执行该转换的条件，而 Action 表示该转换所执行的活动。

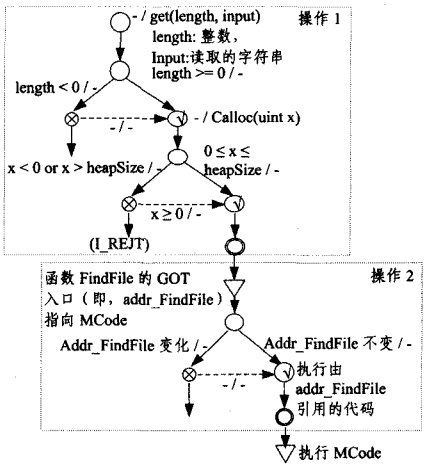

图 6 CFileFind::FindFile()堆溢出脆弱性

在图 6 中，操作 1(覆写 addr_FindFile)是操作 2(执行 MCode)的前置条件，通过上面的传播通道表示，而下面的传播通道(表示为 MCode)则是进入其他恶意操作的前置条件。在程序抽象模型和安全属性断言的基础上，通过调用 SPIN 模型检测器分析脆弱性，可以准确地揭示 MFC 库 CFileFind::FindFile()堆溢出脆弱性的安全违犯情况，并且获得触发脆弱性的执行轨迹，便于准确地定位和分析脆弱性。

结束语 本文提出了一种基于模型检测的二进制程序脆弱性分析框架，设计并实现了相应的脆弱性分析工具原型 MBVA。通过深入地研究二进制程序脆弱性的特点，提出了二进制程序的抽象模型以及基于有限状态自动机的软件脆弱性形式化表示方法。将软件脆弱性建模为一系列基本 FSM (eFSM)的集合。

根据软件脆弱性的形式化模型，为了便于自动化地检测和分析程序中的脆弱性，我们提出了基于事件系统的安全属性表示方法，用于指定推导出的安全断言。在此基础上，提出了基于模型检测的二进制程序脆弱性分析过程和算法，用于验证指定的程序模型是否满足一个安全属性的集合。本算法基于符号模型检测方法提供程序模型的可疑状态分析。在二进制程序脆弱性分析框架的基础上，设计并实现了 MBVA 工具原型，并且通过实验分析 Microsoft MFC 库 CFileFind::FindFile()堆溢出脆弱性，验证了此分析框架的有效性。

然而，对于一个给定的安全属性，正确地构造其形式化声明通常较为复杂。在下一步的工作中，将针对二进制程序模型，研究不同类型安全属性的自动构造方法。另外，对于通用的安全属性和脆弱性模式，如何提供形式化表示并且以自动方式构建其对应关系，也是需要进一步研究的问题。

参 考 文 献

[1] Singh P, Lakhotia A. Static verification of worm and virus behavior in binary executables using model checking[A]// 4th IEEE Information Assurance Workshop[C]. June 2003

[2] Sheyner O, Haines J, Jha S, et al. Automated generation and analysis of attack graphs[A]// Proc. 2002 IEEE Symposium on Security and Privacy[C]. 2002:254-265

[3] Ghosh M C, Simple A. State-based approaches to program-based anomaly detection[J]. ACM Transactions on Information and System Security, 2002, 5(3):203-237

[4] Clarke E, Grumberg O, Long D. Model Checking [M]. MIT Press, 1999

[5] Huth M, Ryan M. Logic in Computer Science, Modelling and Reasoning About Systems[M]. Second Edition. Cambridge University Press, 2004

[6] Ramakrishnan C, Sekar R. Model-based analysis of configuration vulnerabilities[J]. Journal of Computer Security, 2002, 10(1/2):189-209

[7] Chen H, Wagner D. MOPS: an Infrastructure for Examining Security Properties of Software[A]// Proceedings of the ACM Computer and Communications Security (CCS) Conference[C]. November 2002:235-244

[8] Ritchey R, Ammann P. Using model checking to analyze network vulnerabilities[A]// Proceedings of IEEE Symposium on Security and Privacy[C]. May 2000:156-165

[9] McCullough D. Specifications for Multi-level Security and a Hook-Up Property[A]// Poceedings of the 1987 IEEE Symposium on Research in Security and Privacy[C]. IEEE Press, May 1987

[10] Johnson D M, Thayer F J. Security and the Composition of Machines[A]// Proceedings of the Security Foundations Workshop [C]. Franconia, NH, June 1988:72-89

[11] Holzmann G J. The model checker SPIN[J]. IEEE Transactions on Software Engineering, 1997, 23(5):279-295

[12] The SMV System homepage[EB/OL]. URL. http://www.cs.cmu.edu/~modelcheck/smv.html

[13] The LaRS homepage[EB/OL]. URL: http://eis.jpl.nasa.gov/lars/

[14] Microsoft CFileFind:: FindFile Heap Overflow Vulnerability [EB/OL]. URL. http://cve.metre.org/cgi-bin/cvename.cgi?name=CVE-2007-4916

(上接第 109 页)

[11] Han Huaizhong, Shakkottai S, Hollot C V, et al. Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet[J]. IEEE/ACM Transactions on Networking, 2006, 14(6):1260-1270

[12] Melakessou F, Sorger U, Suchanecki Z. MPTCP: Concept of a Flow Control Protocol Based on Multiple Paths for the Next Generation Internet[C]// Proceedings of the 7th International Symposium on Communications and Information Technologies ISCIT'07. Crowne Plaza Hotel, Darling Harbour, Sydney, Australia, October 16

[13] Chen Jiwei, Xu Kaixin, Gerla M. Multipath TCP in Lossy Wireless Environment[C]// IFIP Third Annual Mediterranean Ad Hoc Networking Workshop, (Med-Hoc-Net 2004). June 2004

[14] Radunovi'c B, Gkantsidis C, Gunawardena D. Peter Key Horizon: Balancing TCP over Multiple Paths in Wireless Mesh Network[C]// MobiCom '08: Proceedings of the 14th ACM international conference on mobile computing and networking. 2008: 247-258

[15] Ho T, Karger D R, Medard M, et al. The benefits of coding over routing in a randomized setting[C]// Hideki I. The 2003 IEEE International Symposium on Information Theory. San Jose: IEEE Press, 2003:442-447