基于路径约束的间接跳转目标地址识别

李 丹 王震宇 井 靖 王国好
(信息工程大学 郑州 450002)

摘 要 间接跳转目标地址的识别一直是二进制代码控制流重构的难点之一,其跳转目标一般依赖于程序动态执行时的信息,传统方法无法精确识别。通过对控制流重构技术的研究,提出一种基于路径约束的间接跳转目标地址识别方法,即对于一个间接跳转,在初始控制流图的基础上构建从程序入口点到间接跳转的路径集合,对于每条路径,首先通过数据流分析相关技术得到跳转目标地址关于自由变量的一个表达式,然后对路径约束求解,得到满足约束的自由变量的一组特定解,并以此确定跳转目标表达式的值。通过该方法,每个间接跳转都可以根据路径集合确定跳转目标的地址集合。

关键词 控制流重构,间接跳转,目标地址识别,路径约束,数据流分析
中图法分类号 TP311.5 **文献标识码** A

Recognition of Indirect Jump Targets Based on Trace Constraint

LI Dan WANG Zhen-yu JING Jing WANG Guo-hao
(Information and Engineering University, Zhengzhou 450002, China)

Abstract The recognition of indirect jump targets is one of the most difficult problems for control flow reconstruction from binary all the time. Due to dependency for runtime information, traditional solutions of indirect jump cannot meet the demand of precision. Through research of control flow reconstruction technical, we presented a new method for recognition of indirect jump targets based on trace constraint. For an indirect jump, a trace set from entry point to indirect jump can be constructed from a given initial control flow graph. Then for every element of the trace set, a target address can be determined through trace constraint resolve and target address expression which is constructed with relevant control flow analysis technical. By this means, a set of jump targets can be determined for every indirect jump.

Keywords Control flow reconstruction, Indirect jump, Recognition of target address, Trace constraint, Data flow analysis

二进制代码分析是软件维护、复用、移植、脆弱性分析、恶意代码检测等方面的一个重要方法和手段[1,2]。在二进制代码分析中,控制流重构是关键步骤,当前控制流重构面临的难点之一是间接跳转目标地址的识别[3]。对于间接跳转目标地址的识别,静态分析方法一般无法有效解决;动态分析可以识别,但依赖于程序的测试用例集合,开销大,覆盖率低;而动静相结合的分析方法缺乏一个合理有效的结合点,识别结果不理想。

本文提出了一种基于路径约束的间接跳转目标地址识别方法,即利用从程序入口点到待解决的间接跳转指令的部分路径集合和程序的动态执行信息来识别跳转目标地址。对于一个间接跳转指令,首先构建从程序入口点到该间接跳转指令的路径集合;然后对于每条路经,通过数据流分析得到间接跳转目标地址的表达式,同时对于该路径,综合其在其各个条件分支满足的约束条件,构建其路径约束条件,并对路径约束求解,从而得到满足约束的相关自由变量(也称为符号化变量)的一组特定解;最后将自由变量的值与目标表达式相结合即可确定间接跳转的一个目标地址。根据路径集合的元素个数,每个间接跳转都可以得到一个跳转目标地址的集合。

1 相关研究

当前,针对间接跳转目标地址的识别,比较典型的有以下几种方法。

EEL[4]是一个可执行程序控制流重构工具,它主要使用切片技术来确定间接跳转的目标地址。当跳转目标不能静态确定时,EEL会在程序中插入代码以便在运行时确定。对于某些简单架构,如SPARC架构,EEL可以得到一个相对精确的控制流图,但对于一些复杂架构和编译器,间接跳转目标地址的识别结果不理想。

针对间接跳转,De Sutter等人[5]提出在初始的控制流图中将所有的间接跳转同一个虚拟的未知节点相连接,之后在该控制流图上使用常数传播等策略来证明多数间接跳转目标的不可达性,并删去这些跳转目标。但该方法在跳转目标可达性的判定上不够精确,无法清除许多不可达节点。

Kinder提出了一种基于抽象解释的控制流重构方法[6]。针对间接跳转,该方法依据当前数据流信息,把目标表达式的

李 丹(1985-),男,硕士生,主要研究方向为软件逆向工程,E-mail:ld851221@126.com;王震宇(1969-),男,副教授,硕士生导师,主要研究方向为软件逆向工程、嵌入式系统;井 靖(1980-),女,讲师,主要研究方向为软件逆向工程、嵌入式系统;王国好(1987-),男,硕士生,主要研究方向为软件逆向工程。

所有可能值都作为间接跳转的目标地址，这种处理方式不考虑程序的副作用，很可能会使程序跳转到实际不可能执行到的地方，而且由于引入了过多的目标地址，容易导致内存溢出，使分析无法继续进行；针对该问题，作者又提出了一种改进方案，即依据当前的数据流信息直接对间接跳转目标赋值，使程序跳转到一个确定的地址[7]，该方法只使用程序的一个可能的跳转地址来近似间接跳转的目标，程序覆盖率较低，而且作者没有证明该跳转目标确实是程序实际运行时的一个目标地址。

为了从二进制代码中构建精确的控制流图，California 大学 Davis 分校开发了一个基于 QEMU 的原型系统 FXE (Forced eXecution Engine)[8]，它通过在每个条件跳转的地方强制改变指令指针 IP 的值来进行路径遍历。对于间接跳转，FXE 把经过该间接跳转的所有动态路径的实际跳转目标作为控制流图中间接跳转的跳转目标。事实上，FXE 并未指定任何实际的输入，当需要输入数据时，程序直接从当前内存中取出一个数据，相当于一个随机输入；而且通过强制改变当前指令指针 IP 的值来遍历其它路径很可能会引起程序异常。

通过对上述间接跳转的几种典型处理方法进行分析，我们发现这些方法都存在不足之处，针对间接跳转，当前缺乏一种通用的、可以有效识别跳转目标地址的方法。

2 基于路径约束的间接跳转目标识别

为了精确识别间接跳转目标地址，本文提出了一种基于路径约束的间接跳转目标识别方法。

2.1 系统框架

基于静态反汇编得到的初始控制流图，本文提出了基于路径约束的间接跳转目标地址识别技术，系统框架如图 1 所示。

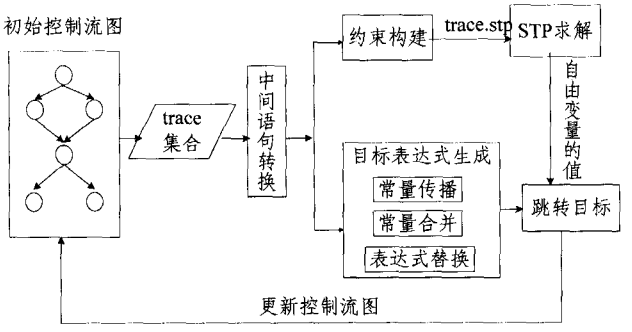
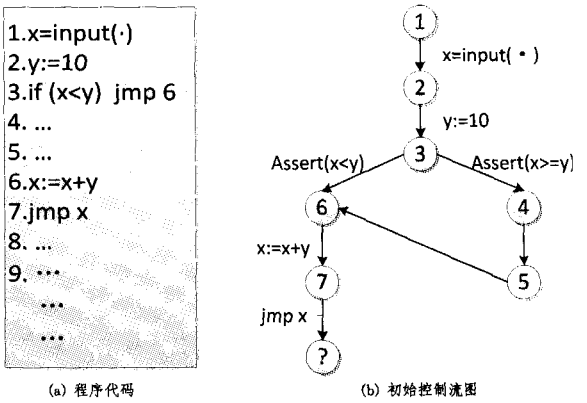


图 1 基于路径约束的间接跳转目标识别系统框架

该框架主要由 5 部分组成：部分 trace 集合产生、中间语句转换、路径约束构建、STP 求解，以及目标表达式生成。第一步是构建从程序入口点到间接跳转的 trace 集合；然后针对每条路经，首先将其转换为中间语言，接下来在中间语言的基础上，通过约束构建和 STP 求解，给出满足该路径约束的自由变量的一组特定解；同时对于该路径，通过常量传播、常量合并、表达式替换等数据流分析技术，获取间接跳转目标的表达式；最后，将目标表达式与自由变量的值相结合，确定间接跳转的一个目标地址。下面结合一个实例，分别对该框架的各部分进行介绍。

程序片段如图 2(a)所示，初始的控制流图如图 2(b)所示。



(a) 程序代码　　(b) 初始控制流图

图 2 间接跳转示例

简单起见，假设每条指令长度为 1，程序代码每行左边的数字表示地址；初始控制流图中“?”表示无法确定的间接跳转目标。在地址 7，jmp x 是一个间接跳转，跳转目标无法确定。

为了识别间接跳转目标地址，首先构建从入口点到间接跳转的部分 trace 集合。

2.2 部分 trace 集合产生

在基于路径约束的间接跳转目标地址识别中，部分路径集合构建是指基于静态分析得到的初始控制流图，构建从入口点到间接分支指令的部分路径集合。

在程序初始控制流图 $G=(N,E,h)$ 中，假设存在若干间接分支指令，那么从程序入口点到间接转移类指令可能存在 n 条部分路径。本质上，这里的部分路径是满足如下约束条件的一个指令执行序列 $\{s_0,s_1,\cdots,s_k\}$：

(i) s_0 是程序入口点的指令，s_k 是间接转移类指令；

(ii) $\forall(0\leqslant i<k),(s_i,s_{i+1})\in E$。

在初始控制流图的基础上，从程序入口点到间接跳转可能存在多条路径，为了完整地构建路径集合，提出了一种基于基本块的深度优先搜索算法(DFBB)。算法的基本思想是：首先在初始控制流图的基础上划分基本块并记录各个基本块的前趋，然后从间接跳转指令所在的基本块开始依次查找当前基本块的前趋；如果前趋个数为 1，则继续进行；如果前趋个数大于 1，则首先保存当前路径信息，并将当前基本块的状态设置为 gray，然后从当前基本块的前趋列表中任选一个(同时将其从前趋列表中删除)进行后向搜索，直到程序入口点所在的基本块。在一条路径搜索完成后，遍历该路径中状态为 gray 的基本块，恢复路径信息，并从前趋列表中选择一个进行后向搜索，同时将该前趋节点从前趋列表中删除，然后判断前趋列表是否为空，如果为空将状态设为 black，直到所有路径中都没有状态为 gray 的基本块，此时可以得到一个 trace 集合。接下来把每条路径中的基本块进行逆序排列，最终可以得到所需的 trace 集合。

在构建 trace 集合时，需要注意对循环的处理。对于路径中存在循环的情况，循环执行次数不同，则得到的路径也不同，因此有必要采取一种合理的策略对循环进行处理。

在汇编指令序列构成的基本块中，循环体是一个特殊的基本块，它的最后一条指令是一个跳转目标地址小于当前指令地址的条件跳转。对于该条件跳转来说，当前面临两条分支：如果条件成立，它的目标地址为该循环体的首地址，程序

会进入循环继续执行；否则程序就会跳出循环。对于跳出循环的情况，循环的处理已经结束；而对于进入循环的情况，当程序执行到条件跳转时，又会面临分支选择。理论上讲，循环体的 $n(n\geqslant 1)$ 次执行都应作为不同的路径。在实际中，循环一般不是无限的，即使是无限循环，也可以通过设置循环次数来进行限定。因此在实现上，对于循环，可以通过设置循环次数上限来进行处理，而且不同的用户根据需求可以设置不同的上限。本文将循环上限设为 2，即把循环体的 1 次和 2 次执行作为不同的路径，不考虑 2 次以上循环的情况。例如对于图 3 所示的指令序列，从地址 k 到 m 为一个循环体，循环体的 1 次执行和 2 次执行对应的 *trace* 分别为 $(1,\cdots,k,\cdots,m,m+1,\cdots)$ 和 $(1,\cdots,k,\cdots,m,k,\cdots,m,m+1,\cdots)$。

```
1.    x:=input(·)
      ...
k.    ...
      ...
      cmp x 5
m.    jb k
m+1.  ...
      ...
```

图 3　指令序列

针对图 2，为了构建 *trace* 集合，按照上文所示方法，首先划分基本块并记录前趋，$BB_1=\{1、2、3\}$，$BB_2=\{4、5\}$，$BB_3=\{6\}$，$BB_4=\{7\}$，然后从 BB_4 开始进行深度优先后向搜索，BB_4 的前趋为 BB_3。由于 BB_3 有两个前趋 BB_1 和 BB_2，分别从两个前趋后向搜索，直到入口点所在基本块 BB_1，最终得到两条不同的路径，其中 $trace1=(BB_1、BB_3、BB_4)$，$trace2=(BB_1、BB_2、BB_3、BB_4)$。

在初始控制流图中，对于每个待处理的间接转移类指令，通过基于基本块的逆向深度优先遍历，可以构建一个部分路径集合。

2.3　中间语句转换

接下来，为了构建路径约束条件，需要对每条路径进行分析。一般情况下，二进制分析可以分为两类，一类是直接基于汇编指令进行分析；一类是将汇编指令转换为中间语言，然后基于该中间语言进行分析。由于现代典型架构处理器的指令集一般都比较庞大，指令种类繁多，语义结构复杂，而基于汇编构建约束条件时，需要对每条指令的语义进行分析，此外还要考虑每条指令对程序状态的影响，难度较大，因此本文采取后者，即首先将汇编转换为中间语言，并在中间语言中明确标明指令的语义和寄存器标志位信息，然后基于中间语言进行分析。

当前在二进制分析领域，中间语言的种类繁多，它们是不同分析人员针对不同目的而设计的。依据本文使用中间语言的目的，在现有中间语言的基础上重新设计一种合适的中间语言。在实现上，可以选取 BIL 作为分析使用的中间语言。BIL 是 BAP[9] 中用的中间语言，在设计上，它很好地抓住了汇编语言的特征，转换后的中间语义与汇编基本等价，而且 BIL 通用性好，表达性强，明确指出了指令对寄存器标志位的影响，符合本文应用的需求；此外 BAP 和 BitBlaze[10] 都支持将不同架构和平台的汇编转换为 BIL 中间语言，有助于实现中间语言的转换。

一般情况下，一条汇编指令对应若干条 BIL 中间语句，这些中间语句包含对应汇编的具体操作以及指令对各个标志位的影响，两者在语义上是等价的，虽然从形式上看中间语言比较复杂，但其明确了指令的语义及其对相关状态标志位的影响，很大程度上降低了后续路径约束构建的难度。

在该示例中，为了简单起见，程序代码使用了一种与 BIL 类似的中间语言，因此中间语言转换这一步可以省略。需要说明的是，在实际工作中，我们面对的是汇编指令，而且已经实现了汇编到中间语言的转换。

接下来，对于每条路径，我们需要构建其路径约束条件。

2.4　路径约束构建

在基于路径约束的间接跳转目标识别框架中，路径约束的产生是关键，对于从程序入口点到间接跳转指令的路径集合，需要根据每条路径的指令流信息构建其路径约束条件。

在路径转换后的 BIL 中间语言中，每个条件跳转的 condition 被定义为 Bool 类型的变量 post_xx，其中 xx 是一个数字编号，不同条件跳转的 condition 对应的 post 的编号不同。对于一条确定的路径，为了构建其路径约束条件，对于每个条件跳转，首先需要从程序的入口点开始，通过数据流分析，求出当前 post_xx 关于自由变量的表达式，然后根据当前路径的分支选择情况判断在该路径上 condition 的值，最后令 post_$xx=0$ 或 1 表示路径在当前条件下跳转需满足的约束条件。

对于一条路径，将其所有条件跳转节点的约束条件合并起来可得到最终的路径约束条件(trace constraint condition)。路径约束生成算法描述如下：

```
Algorithm  Generate TCC 生成 TCC
Input: trace. il // trace. il 表示路径的中间语言文件
    Output: trace. stp  // trace. stp 表示路径的约束条件
begin
1. * pfile := fopen("trace. il","r")
2. post_queue := ? //post 队列,初始为空
3. while (!pfile) //判断文件指针是否为空
4.   post_queue := ENQUEUE(post_queue,post_xx)
     //扫描整个文件,将所有 post 节点入队
5.   return T
6. end while
7. start :=get_entry_node()
   //获取路径的入口地址
8. if (!EMPTY(post_queue)) then
9.   currentt_post := FRONT(post_queue)
     //取队头元素
10. ASSERT(current_post=get_post_bool(current_post))
      //获取第一个条件跳转对应的约束条件
11. end if
12. while (!EMPTY(post_queue))
13.   next_post :=FRONT(post_queue)
14.   ASSERT(next_post=get_post_bool(next_post))
15.   current_post :=next_post
16.   jmp 12
17. end while
end
```

对于每条路径，通过该算法可以生成一个路径约束条件。

下面按照约束条件的构建方法，分别对两条路径进行处理。

对于 trace1，该路径上自由变量为 x，沿该路经的条件分支需满足 $x<y$，按照 STP 的输入规范，构建路径约束条件如下：

```
x:BITVECTOR(32);
ASSERT(
LET y=0hex0000000A
IN
BVLT(x,y));
QUERY(FALSE);
COUNTEREXAMPLE;
```

其中 x 是一个 32 位的位向量，也是一个自由变量；LET y=0hex0000000A IN 表示引入一个变量 y，同时对其赋初值；BVLT 是 STP 的算数运算函数，BVLT(x,y)表示 $x<y$；ASSERT(*)表示令括号内的约束条件成立；QUERY 是对约束条件进行查询，这里 QUERY(FALSE)表示查询约束条件是否恒为假，如果对于自由变量 x 的任意值，约束条件恒为假，STP 会给出 Valid，否则 STP 会给出 Invalid，并给出一个令约束条件为真的自由变量的值，这里给出的是 x 的值；COUNTEREXAMPLE 便是 STP 给定的满足约束条件的自由变量的解；同理对于 trace2，我们也可以构建这样一个路径约束条件。

接下来，利用约束求解器，如 STP，可以对路径约束条件进行求解。

2.5 STP 求解

在基于路径约束的间接目标地址识别中，STP 求解主要是针对部分路径的约束条件，求解满足条件的输入，使得程序的动态执行路径可以覆盖该部分。

STP(Simple Theorem Prover)[11]是一个约束求解器，或称为决策程序，它属于一个 SMT(Satisfiability Modulo Theories)求解器。SMT 求解器是求解 SMT 问题的自动化工具，它处理的对象是包含一些特定理论的一阶逻辑公式，典型的理论包括固定长度的位向量、数组等，目前 SMT 求解器已经成为许多程序分析和验证工具的基础组件。

STP 可以接受程序分析工具、测试用例产生工具、模型检测器、漏洞挖掘工具等产生的约束条件，并给出一个满足约束的特定的解。它具有标准的输入规范和特定的输入语言，在其输入文件中，首先是对自由变量的声明，然后是约束条件，最后是约束查询并给出一个满足约束的解。

对于路径约束条件，通过 STP 求解，可以得到满足约束条件的自由变量的一组特定解，它一般是一个特定的输入，该输入可以使程序沿该路径执行。

对于 trace1，STP 查询给出了判定结果 Invalid，并给出了自由变量 x 的解：x=0hex00000004；同理对于 trace2，STP 给出的判定结果为 Invalid，自由变量 x 的解为 x=0hex0000000B。

接下来，对于每条路径，我们需要求出其目标地址表达式。

2.6 目标表达式生成

对于一条特定的路径，通过数据流分析的相关技术，可以得到间接跳转的目标地址关于自由变量的一个表达式。相对于在整个二进制程序上进行的数据流分析，基于路径的数据流分析比较简单，只需在路径上使用常量合并、常量传播和表达式替换便可得到间接跳转目标的一个表达式。

在数据流分析中，常量合并是指将程序中的常数表达式之值先行算出，而不必生成用于计算该常数表达式的代码。如可将表达式 $a+2*3$ 翻译成 $a+6$。常量传播，是指如果在一段程序范围之内，变量的值保持不变，那么对该变量的引用可以替换为对其值的直接引用，这种常量传播一直延续到该变量被重新赋值时为止[12]。

在基于路径的数据流分析中，表达式替换是获取目标表达式的关键。在程序中，寄存器和部分的存储单元在语句执行的过程中起临时存储和转移数据的作用，通过跟踪寄存器和这部分存储单元中数据的流向，累积它们数据运算的效果，并将最终的运算结果体现到间接跳转目标中[13]。

在一条特定的路径上，局部变量一般是常数或者自由变量组成的表达式，通过上述 3 种策略，将这些中间数据的运算效果直接体现在跳转目标表达式中，最终可以得到间接跳转目标关于自由变量的表达式[14]。

对于 trace1，根据数据流分析技术生成该路径上间接跳转目标地址的表达式。该路径上，x 为自由变量，在地址 2 处引入了局部变量 y :=10，在 y 被重新定义之前用 10 替换 y；在地址 6，x 被重新定义为 $x+10$；在地址 7，根据表达式替换原则，用 $x+10$ 替换 x，最后得到跳转目标地址表达式，即 $x+10$。同理对于 trace2，其目标地址表达式也为 $x+10$。

最后，对于一条路径，将约束求解得到自由变量的值和间接跳转目标地址表达式结合，即可得到满足该路径约束的一个间接跳转目标地址。因此对于图 2 中所示的间接跳转，最终可以得到两个目标地址 x :=14 和 x :=22。

接下来，利用这两个目标地址对初始控制流图进行更新，更新后的控制流图如图 4 所示。

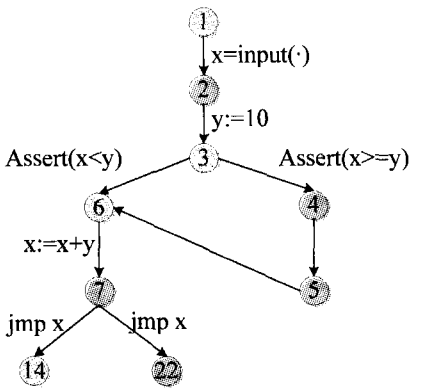


图 4 更新后的控制流图

至此，基于路径约束的间接跳转目标地址识别已经完成。在初始的控制流图中，可能存在多个间接跳转指令，利用本文提出的方法对所有的间接跳转指令进行处理可以有效提高控制流图的精确性。

3 实验验证

控制流重构中，当前比较有代表性的是 Kinder 提出的将数据流和控制流相结合的基于抽象解释的控制流重构。为了验证本文提出的基于路径约束的间接跳转目标地址识别方法的可行性和有效性，首先利用基于抽象解释的控制流重构方

法识别图 2 中间接跳转目标地址，识别结果如图 5 所示。

其中图 5(a)所示的控制流图是按照文献[6]介绍的基于抽象解释的控制流图重构方法得到的控制流图，对于间接跳转，该方法把跳转目标表达式的所有值都作为间接跳转的目标地址，控制流图中的虚线表示所有可能的边；在文献[7]中，作者对上述方法进行了改进，对于间接跳转，直接对跳转目标赋值，按照此方法，在地址 7，令跳转目标 $x=14$，控制流图如图 5(b)所示。

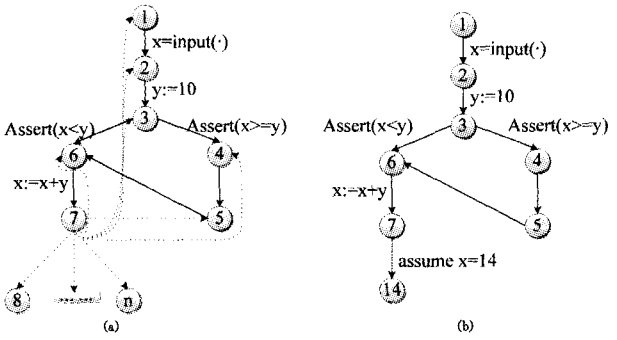


图 5　基于抽象解释的控制流图

通过对比发现，对于间接跳转目标地址的识别，基于抽象解释的控制流重构引入了过多的目标地址，得到的控制流图不精确；改进后的方法虽然不会引入错误的跳转目标地址，但对于每个间接跳转只能得到一个目标地址，覆盖率低。而本文提出的基于路径约束的间接跳转目标识别方法可以依据从入口点到间接跳转的路径集合识别出多个目标地址，而且可以保证目标地址的正确性。

结束语　本文提出的基于路径约束的间接跳转目标识别方法充分利用了初始控制流图中隐含的约束条件和程序的动态执行信息对间接跳转的目标地址进行识别，相对于单纯的静态或动态分析方法，该方法较好地结合了两者的优点，可以有效识别出间接跳转的目标地址。

针对间接跳转的目标地址识别，当前缺乏一种通用的且能够精确识别其跳转目标地址的方法。本文提出的方法虽然能够准确识别出一个跳转目标地址的集合，但仍存在一些不足。首先，利用本文提出的方法得到的间接跳转目标地址集合是实际目标地址集合的一个子集，其虽然可以保证正确性，但在覆盖率上还有待提高，尤其是对于 switch-case 的跳转表形式。其次，该方法在工程实现上有所欠缺。下一步，计划对该方法进一步优化，充分利用程序的数据流信息和动态运行信息，在保证间接跳转目标地址识别正确性的同时，尽可能提高完整性。

参 考 文 献

[1] Balakrishnan G. WYSINWYX: What you see is not what you execute[D]. Wisconsin: University of Wisconsin, 2007

[2] Cifuentes C. Reverse Compilation Techniques [D]. Queensland: Queensland University of Technology, 1994

[3] Bardin S, Herrmann P, Vedrine F. Refinement-Based CFG Reconstruction from Unstructured Program [C] // Verification Model Checking and Abstract Interpretation (VMCAI), 2011. Berlin: LNCS 6538, 2011: 54-69

[4] Larus J R, Schnarr E. EEL: Machine-Independent Executable Editing[C]// Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. 1995: 291-300

[5] De Sutter B, De Bus B, De Bosschere K. Link-time binary rewriting techniques for program compaction[J]. ACM Trans. Program. Lang. Syst., 2005, 27(5): 882-945

[6] Kinder J, Zuleger F, Veith H. An Abstract Interpretation-Based Framework for Control Flow Reconstruction from Binaries[C]// Verification Model Checking and Abstract Interpretation (VMCAI), 2009. Berlin: LNCS 5043, 2009: 214-228

[7] Kinder J, Kravchenko D. Alternating Control Flow Reconstruction[C]// Verification Model Checking and Abstract Interpretation (VMCAI), 2012. Berlin: LNCS 7148, 2012: 267-282

[8] Xu L, Sun F, Su Z. Constructing precise control flow graphs from binaries[R]. Tech. Rep. University of California, Davis, 2009

[9] Binary analysis platform (BAP) [OL]. http://bap.ece.cmu.edu, 2012-11-25

[10] Song D, Brumley D, Yin Heng, et al. BitBlaze: A New Approach to Computer Security via Binary Analysis[C]// Intelligent Computing and Integrated Systems Security (ICISS), 2008. Berlin: LNCS 5352, 2008: 1-25

[11] STP: A Decision Procedure for Bitvectors and Arrays[OL]. http://people.csail.mit.edu/vganesh/STP_files/stp.html, 2012-11-01

[12] 方霞. 代码逆向分析中的语句恢复与算法识别技术研究[D]. 郑州：信息工程大学，2009

[13] 殷文建. 面向 ARM 体系结构的代码逆向分析关键技术研究[D]. 郑州：信息工程大学，2009

[14] Schwartz E J, Avgerinos T, Brumely D. All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution[C] // IEEE Symposium on Security and Privacy, 2010. Pittsburgh: DOI 10.1109/SP, 2010, 26

(上接第 282 页)

[8] 黎华，王周敬. 基于直觉模糊集的多属性决策问题[J]. 昆明理工大学学报：理工版，2008

[9] 张娓娓，范训礼，等. 基于模糊理论的 P2P 流媒体节点选择算法[J]. 计算机工程，2009

[10] 张治斌，冯文峰，黄永峰. 基于 Gossip 的自适应成员关系管理协议[J]. 计算机应用，2009(11)：2932-2935

[11] 叶枫，张思发. 基于 Gossip 协议的 P2P 流媒体直播系统的研究[J]. 计算机与数字工程，2009，37(5)：89-93

[12] 伍红华. 基于 Gossip 协议的 P2P 内容分发系统模型[J]. 湖北教育学院学报，2007，24(8)：29-31

[13] 袁雪萍，周芳，陈璐. 基于 Gossip 协议的 P2P 流媒体算法优化[J]. 计算机与现代化，2010(10)：130-141

[14] 郭良敏，杨寿保，郭磊涛，等. P2P 网络中基于区域划分的超级节点选取机制[J]. 小型微型计算机系统，2008，29(2)：208-212