基于特征值的模板化库函数识别①

汪 玮，陈凯明

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)

摘 要：逆编译是编译的逆过程，目的是将可执行的二进制机器代码变换为功能等价的高级语言代码，它在监控恶意代码，挖掘软件漏洞的工作中起着重要的作用。而在面向对象语言的逆编译过程中，模板化库函数识别的难度和复杂性非常大。通过分析现有的 C++库函数识别方法，针对库函数模板识别中特征值冲突的关键问题，提出一种改进的特征值构造方法，它能够更完整的保存库函数信息，降低特征值冲突出现的概率，并通过识别部分 C++标准模板库函数验证了该算法。

关键词：逆编译; 模板; 库函数; 识别;

Recognition of Template Library Functions Based on Features

WANG Wei, CHEN Kai-Ming

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: The decompilation decode binary instructions and data into high level source code against compilation. It plays an important part in finding malware and discovering software vulnerability. But the recognition of template library functions is very difficult in decompilation of object-oriented language. This paper analyzes the existing algorithm of library function recognition and aims at the key problem of feature conflicts to introduce an improved algorithm based on feature-recognition. This algorithm can store much more library function informations to avoid feature conflicts and has been verified in some experiments about C++ STL recognition.

Key words: decompilation; template; library function; recognition

逆编译是编译的逆过程，目的是将可执行的二进制机器代码变换为功能等价的高级语言代码，它在监控恶意代码，挖掘软件漏洞的工作中起着重要作用[1]。库函数识别是逆编译的重要过程，成功的库函数识别能为以后的控制流、数据流恢复打下非常重要的基础，直接影响逆编译结果的可读性。C++ 语言是一门使用非常广泛的面向对象语言，然而对其逆编译的研究并没有取得很大的成果。文献[3]已经对面向对象语言的特点和 C++模板库函数与普通库函数之间的区别做了详细的分析。简言之，C++库函数不同于 C 库函数，它主要是以模板的形式表现，所以研究如何能够正确识别模板化库函数在 C++逆编译研究中具有很大的价值。模板化库函数的实际代码在编译阶段才真正生成，根据模板参数的不同生成的代码也不同。所以针对 C 库函数生成的可执行代码相对固定特点提出的库函数识别方法不再适用于 C++模板化库函数识别。

模板化库函数识别的作用如图 1 所示，它能够从可执行程序的反汇编代码中识别模板化库函数，例如图中标示的 deque_front、deque_popfront、deque_pushback 函数。其中 call sub_401A20,call sub_401170 为用户自定义函数调用，不属于库函数识别工作。

目前主流的逆编译软件在无符号信息的基础上都不能实现 C++模板库函数识别[2]，并且针对 C++库函数识别方法研究的论文很少，国外文献也没有对模板化库函数识别方法作出相关研究。国内给出具体实现算法的仅有文献[3,4]

① 收稿时间:2011-11-15;收到修改稿时间:2011-12-15

文献[3]根据同一模板代码结合同一类型参数所实体化结果一致的特点，提出了一种识别 C++模板化库函数的方法。然而当模板库函数与用户自定义的类型参数结合使用的时候，该方法就无法识别库函数。文献[4]针对同一模板与不同参数类型生成的库函数具有一定的共同特征实现了一种提取模板化库函数特征值的方法。然而该方法存在了一定的不足，简单的通过通配符”****”设置特征值将导致部分库函数信息丢失，从而提高了特征值冲突的概率。

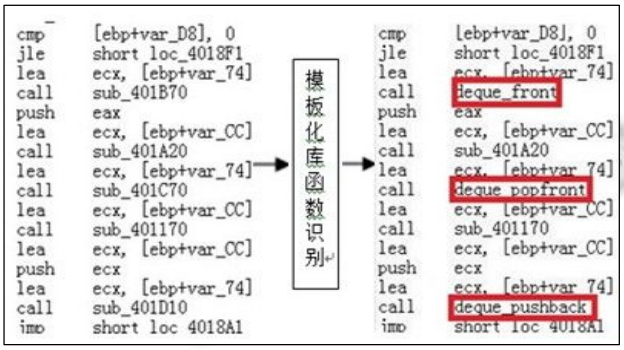

图 1 库函数识别

本文将提出一种模板化库函数特征值提取算法，通过保存更完整的库函数信息，降低特征值冲突，实现从反汇编代码中识别 C++模板化库函数的功能。

1 算法分析与设计

特征值识别法是逆编译中库函数识别常采用的方法，即通过对库函数的分析总结得到其特征，然后根据这些特征来匹配待识别的中间代码。特征值识别法的难点就在于如何定义特征值与解决特征值之间的冲突问题[5]。

现有特征值提取算法以指令为基础单元，只要指令之间有不同就会使用通配符替代特征值中的对应位置。该方法的优点是特征值构成简单，提取算法速度快，能够适用于大部分模板库函数，但是该方法会出现以下问题。

如果模板函数体太过短小且与类型参数关系太过密切时，提取的特征值长度小，且通配符占有比例高(以下简称之为非健壮特征值)，会导致误识别的出现。表 1 中列出了一条非健壮特征值与一条健壮特征值的对比例子。

通过识别算法，任何满足该条健壮特征值的代码段也满足此条非健壮特征值，从而导致误识别，库函数误识别将直接影响逆编译结果的可读性，具有非常大的危害。例如标准算法 find 函数就会出现以上情况，被提取出非健壮特征值，而导致误识别的出现。

表 1 非健壮特征值

一个非健壮特征值	待识别目标代码	一个健壮特征值
指令 a	指令 a	指令 a
通配符****	指令 a0	指令 a0
指令 b	指令 b0	指令 b0
当通配符匹配目标代码的指令 a0、b0、c0、a1、b1、d0、e0 时，则成功识别	指令 c0	指令 c0
	指令 a1	通配符****
	指令 d0	指令 d0
	指令 e0	指令 e0
	指令 b	指令 b

通过分析，出现非健壮特征值的原因是通配符****丢失了原库函数该位置部分的信息，这样做的优点是特征值形式及提取算法简单，但却提高特征值冲突出现的概率，因此我们需要适当的增加特征值的复杂度。

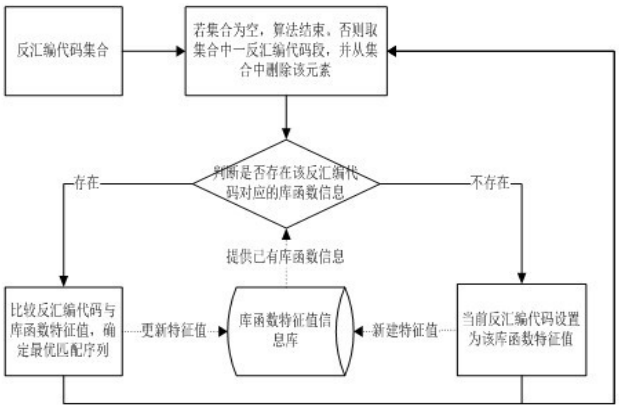

图 2 特征值提取流程

据此我们提出一种新的特征值形式，将指令做更细粒度的划分：

特征值由多条指令构成，指令的形式分为三种：汇编指令，类汇编指令，通配符。

其中汇编指令形式为“操作码 操作数”，表示库函数在不同上下文的目标代码中该位置指令不发生变化。

类汇编指令形式为“操作码 **”，表示库函数在不同上下文的目标代码中该位置指令的操作码不发生变化，仅有操作数发生变化。

通配符的形式为“****”，表示库函数在不同上下文的目标代码中该位置指令不固定，没有规律可循。

根据以上特征值定义，我们提出了以下特征值提取流程，并给予适当的说明。

反汇编代码集合作为整个特征值提取的输入，集合元素是由模板库函数在相同编译环境下不同上下文所生成的可执行代码反汇编得到的。

在介绍反汇编代码与特征值之间的最优匹配序列之前，我们首先定义两个概念：匹配序列与指令相似度。

定义 1. 匹配序列指一个反汇编代码指令与特征值指令之间的一个特殊的对应关系序列，该序列满足以下条件：

反汇编代码中的指令可以对应于特征值的一条指令，或不对应任何指令。

特征值中的指令也可以对应于反汇编代码中的一条指令，或不对应任何指令。

对应关系不能出现交叉，即反汇编代码中的第一条指令对应了特征值中的第二条指令，那么就不允许反汇编代码中的其他指令再对应特征值中的第一条指令。例如表 2 列出的两个具有 10 条指令的代码段之间的部分匹配关系。

表 2　合法匹配与非法匹配

合法匹配	非法匹配
(1,1)(2,2)(3,3)(5,5)(10,10)	(1,1)(2,1)(3,3)(4,4)(5,5)
(1,2)(2,4)(3,6)(4,8)(5,10)	(1,2)(2,1)(3,3)(4,4)(5,5)

定义 2. 指令相似度即一个反应两条指令的相似程度的数值，它分三种情况。当两条指令完全一样，则相似度为 1。当两条指令操作码不同时，相似度为 0。而当两条指令操作码相同，操作数不同的时候，相似度为 X。这里的 X 为自定义的常量，属于区间（0,1）表示当指令操作码相同，指令操作数不同时，指令仍然具有一定的相似度 X。此时我们通过查找一条指令相似度总和最大的匹配序列来提取特征值。根据实验我们发现 X 取值接近 1 时，匹配序列相似度总和最大的序列，其操作码序列基本相同; 而 X 取值接近 0 时，匹配序列相似度总和最大的序列，其指令序列基本相同。而通过现有方法提取出非健壮特征值的库函数在不同上下文生成可执行文件的指令序列变化大，所以我们可以设置较大的 X 来提取该类库函数的特征值。而现有方法可以提取出健壮特征值的库函数，我们则设置较小的 X 或者设置 X=0 来提取该类库函数。

反汇编代码与特征值之间的最优匹配序列指的就是所有匹配序列中相似度总和最大的序列。

在生成了最优匹配序列之后，我们根据该序列，更新原有的特征值。当反汇编集合为空时，即原先集合内所有代码都参与了特征值提取的过程，特征值就完成了所有的更新。下文将详细叙述相关算法的实现。

2　算法实现

2.1 最优匹配序列生成算法

通过指令相似度构成矩阵，由矩阵计算路径，生成最优匹配序列

矩阵的大小为 old_code_length*new_code_length，即已经提取的特征值语句条数与待提取特征值汇编指令条数的乘积。为方便计算，指令都采用数组的计数方式从第 0 条开始计数。

表 3　矩阵构造

已提取的特征值	待提取特征汇编代码
push　ebp mov　ebp，esp sub　esp，18h mov　** push　eax lea　** **** push　edx call　** add　** ****	push　ebp mov　ebp，esp sub　esp，18h push　ecx lea　ecx，[ebp+arg_C] push　ecx lea　edx，[ebp+var_8] push　edx call　sub_403A40 add　esp，8

矩阵	
	1 ,0 ,0 ,0.3 ,0 ,0.3 ,0 ,0.3 ,0 ,0
	0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0
	0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0
	0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0
	0.3 ,0 ,0 ,0.3 ,0 ,0.3 ,0 ,0.3 ,0 ,0
	0 ,0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0
	1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1
	0.3 ,0 ,0 ,0.3 ,0 ,0.3 ,0 ,1 ,0 ,0
	0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0
	0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1
	1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1

矩阵元素的内容为 old_code,new_code 对应的位置指令的匹配权值，如第 i 行第 j 列表示 old_code 的第 i 条指令与 new_code 的第 j 条指令的相似度。

如表 3 给出一个构造矩阵的例子，该例中 X=0.3（本例中 X 的选择没有实际意义，选取实值仅为便于表示最优匹配序列生成的过程）。根据生成的矩阵生成该矩阵从左上角到右下角的右下行最大权路径，该算法类似于有

向图的最大权路径寻找算法，在此就不赘述了。

由上例的矩阵可以得到右下行最大权路径有：(0,0) (1,1) (2,2) (4,3) (5,4) (6,5) (7,7) (8,8) (10,9)

2.2 特征值更新算法

输入：右下行最大权路径 path={(x1,y1)(x2，y2)….}，old_code（已经提取的特征值）,new_code（待提取特征值的汇编代码）。

输出：更新过的 old_code，即新的特征值

令：Sa={X1,X2…Xa}为 old_code 的代码序列，a 为 old_code 代码序列的长度。Sb={Y1,Y2…Yb}为 new_code 的代码序列，b 为 new_code 代码序列的长度。对每个 path 中的（x,y）都建立匹配关系[Xx,Yy],除此之外添加一个对应关系[X(a+1),Y(b+1)]

算法流程如下

(1) 令 i=j=1

Repeat

if Xi 与 Yj 已经建立了匹配关系

i++,j++

Else if Xi 与 Sb 中某个块建立了匹配关系

在 X(i-1)与 Xi 间插入一个空块 k 到 Sa 中，并建立匹配关系[k,Yj],j++

Else if Yj 与 Sa 中某个块建立了匹配关系

在 Y(j-1)与 Yj 间插入一个空块 k 到 Sb 中，并建立匹配关系[Xi,k],i++

Else

建立匹配关系[Xi,Yj]

End if

Until i>a And j>b

此时 Sa,Sb 已经建立了内部代码序列一一对应的关系。

(2) 遍历 Sa

if Xi=Yi

else if Xi 的操作码！=Yi 的操作码，Xi 置为****

else if Xi 的操作数！=Yi 的操作数，Xi 的操作数置为**

遍历结束

(3) 再遍历 Sa，把连续的“****”合并为一个“****”

至此 Sa 中就是 old_code 的更新版。

储存 Sa 作为该模板的特征代码，待以后使用。

继续 3.1 中例子实现特征值更新如表 4：

表 4 特征值更新

Sa,Sb 一一对应关系	更新 Sa	合并“****”
（0,0）	push ebp	push ebp
（1,1）	mov ebp，esp	mov ebp，esp
（2,2）	sub esp，18h	sub esp，18h
（3,k）	****	****
（4,3）	push eax	push eax
（5,4）	lea **	lea **
（6,5）	****	****
（k,6）	****	push edx
（7,7）	push edx	call **
（8,8）	call **	add **
（9,k）	add **	****
（10,9）	****	
（11,10）		

2.3 特征值识别算法

当模板库函数的特征值提取完毕以后，就可以对未知库函数进行识别了。要识别未知库函数，首先要定义未知库函数的指令与特征值指令的匹配关系。

特征值指令有三种形式，所以要考虑三种匹配关系。

未知指令与汇编指令匹配的条件是两条语句是完全相同;

未知指令和类汇编指令匹配的条件是两条语句的操作码是相同的，操作数可以不同，因为类汇编语句的操作数是符号“**”;

未知指令与通配符则在任何情况下都是匹配的。

当一段未知汇编代码与特征值每条指令都匹配的时候，则该汇编代码与该特征值匹配，即该汇编代码属于该特征值对应的模板库函数。

有了匹配的定义，那么现在的库函数识别算法就可以定义成为一种求解动态规划的问题，具体的问题就是如何将未知函数汇编代码中的 0 条和多条语句匹配成特征值的通配符“****”，使得剩余的代码能够完全匹配于特征值中的汇编与类汇编语句。该规划问题可以通过枚举来解决。

表 5 显示了未知代码识别的动态规划结果。如上表该汇编代码能匹配该特征值，因此该汇编代码属于该模板库函数，通过该算法可以识别未知汇编代码是哪种模板化库函数，并返回模板化库函数名。

表 5　动态规划结果

汇编代码	规划所得结果	特征值
push ebp	push ebp	push ebp
mov ebp，esp	mov ebp，esp	mov ebp，esp
sub esp，18h	sub esp，18h	sub esp，18h
push eax	push eax	****
lea ecx，[ebp+arg_C]	lea ecx，[ebp+arg_C]	
push ecx	此处 2 行对应****	
lea edx，[ebp+var_8]	push ecx	push eax
push edx	lea edx，[ebp+var_8]	lea **
call sub_403A40	此处 0 行对应****	****
add esp，8	push edx	push edx
	call sub_403A40	call **
	add esp，8	add **
	此处 0 行对应****	****

3　实验与结果

操作系统：windows 7; 编译环境：VS2008（关闭内联优化）；反汇编软件：IDA pro。

我们以 C++标准模板库中使用率非常高的 24 个库函数作为识别对象，表 6 列出最后识别结果。

表 6　实验结果

库函数模板	文献[4]算法		本文算法	
	正确	误报	正确	误报
Deque_deque()	Y	N	Y	N
Deque_front()	Y	Y	Y	N
Deque_back()	Y	Y	Y	N
Deque_empty()	Y	N	Y	N
Deque_pushfront()	Y	Y	Y	N
Deque_popfront()	Y	Y	Y	N
Deque_erase()	Y	N	Y	N
list_list()	Y	N	Y	N
List_front()	Y	Y	Y	N
List_back()	Y	Y	Y	N
List_empty()	Y	N	Y	N
List_pushfront()	Y	N	Y	N
List_popfront()	Y	N	Y	N
List_erase()	Y	N	Y	N
Vector_vector()	Y	N	Y	N
Vector_front()	Y	Y	Y	N
Vector_back()	Y	Y	Y	N
Vector_empty()	Y	N	Y	N
Vector_pushback()	Y	N	Y	N
Vector_popback()	Y	N	Y	N
Vector_erase()	Y	N	Y	N
Find()	Y	Y	Y	N
For_each()	Y	N	Y	N
Stable_sort()	Y	N	Y	N

(表中正确栏表示只要是属于该模板的库函数都会被识别，误报栏表示该模板特征值会把非该模板的库函数识别成该模板)

4　总结

通过实验，可以看到该方法能够克服现有算法误识别的问题，正确的识别出了模板化库函数，同时因为模板化库函数是库函数的特例，因此该方法同样适用于普通的非模板库函数以及 c 库函数。然而当两个模板库函数的功能结构非常相似情况下（甚至部分库函数生成的可执行代码完全相同），通过以上方法提取的特征值也会出现完全相同的情况，有两种处理方法。方法一：该方法是在提取特征值的过程中，通过文献[6]中提到提取辅助特征值的方法解决此类问题，该方法适用于模板库函数里包含有调用子程序的语句，具体实现请参照该文献。方法二：通过实验发现在 Visual Studio 系列的编译工具下，可执行代码在调用类函数之前，会有传递 this 指针的操作，根据这一特点如果出现冲突的库函数特征值属于某个模板类的时候，我们可以在模板识别冲突发生之后，跟踪当前函数的 this 指针，找出和该函数属于同一 this 指针的其他函数，从其他函数判断该类名，由于目前没有发现同一个类中发生如上冲突，所以可以根据类信息找到该模板库函数的信息，本文实验就是采用该方法处理了部分特征值冲突问题。

参考文献

1 Van Emmerik M. Static Single Assignment for Decompilation. Quessnsland University of Technology,2007.

2 陈耿标.反编译器 C-Decompiler 关键技术的研究和实现.上海交通大学,2010.

3 胡政.C++逆编译中库函数识别研究.计算机工程与应用,2006,142(3):66–68.

4 胡政.C++逆编译中模板库函数识别研究.中国科学技术大学,2006.

5 刘宗田.C 语言反编译系统 DECLER.微电子学与计算机,1997;14(5):1–3.

6 陈福安,刘宗田.8086C 反编译系统中库函数识别技术及其实现.小型微型计算机系统,1991;12(11):33–40.