doi:10.3969/j.issn.1000-436x.2013.09.021

基于软件执行轨迹差异比对的关键函数定位技术研究

康绯[1]，王乾[2]，肖亚南[1]，黄荷洁[1]
(1. 信息工程大学 数学工程与先进计算国家重点实验室，河南 郑州 450001；2. 中国北方电子设备研究所，北京 100191)

摘 要：关键函数是指应用软件在某个运行阶段发挥着关键作用的核心功能函数。对软件中的关键函数进行快速定位是提高逆向分析效率的有效手段。目前，在软件逆向工程领域对关键函数进行定位大多是利用人工分析的方法。利用动态二进制插桩技术，提出了一种切实可行的基于软件执行轨迹差异的关键函数自动定位方法。当软件具有 2 类不同的输入，分别触发、不触发关键函数时，该方法能够快速、准确地识别关键函数。

关键词：关键函数；软件执行轨迹；动态二进制插桩
中图分类号：TP309 **文献标识码：**A **文章编号：**1000-436X(2013)09-0177-08

Research on key functions locating technique based on software execution trace difference comparison

KANG Fei[1], WANG Qian[2], XIAO Ya-nan[1], HUANG He-jie[1]
(1.State Key Laboratory of Mathematical Engineering and Advanced Computing, Information Engineering University, Zhengzhou 450001, China;
2. Institute of North Electronic Equipment, Beijing 10091, China)

Abstract: Key functions are the core functions which play vital es in certain run phase of application software. The quick locating of key functions is a valid method to improve the efficiency of software reverse analysis. In the field of software reverse engineering, locating key functions is mostly based on l analysis. Dynamic binary Instrumentation (DBI) techniques were employed to present a practicabletechnique to automatically locating the key functions based on software execution trace difference.Key functions can be quickly and precisely located when key functions can be triggered or not by two different kinds of software inputs.

Key words: locating function; software execution trace; dynamic binary instrumentation

1 引言

“关键函数”是指应用软件中在某个运行阶段发挥着关键作用的核心功能函数，是软件逆向过程中需要重点进行逆向分析的函数。“关键函数”是与软件逆向目标紧密相关的一个概念。软件逆向目标不同，关键函数的性质也不一样。例如，分析加解密软件时关键函数为核心加解密函数；分析商业软件版权保护机制时关键函数为软件注册验证函数等。

“关键函数定位”是指综合利用反汇编、软件调试、二进制分析等技术手段，通过分析目标软件的静态二进制特征和动态行为特征，进而确定关键函数在软件二进制代码中位置范围的过程。对目标关键函数进行快速定位可以有效地缩小逆向分析的范围、降低逆向分析的复杂度，从而提高软件逆向工作的效率。

2 国内外相关技术研究与发展现状

在实际的软件逆向工程中，由于逆向目标以及在逆向过程中遇到的问题是千差万别的，所以使用静态反汇编工具和动态调试器进行人工逆向分析

收稿日期：2013-05-02；修回日期：2013-06-28
基金项目：国家保密局科研基金资助项目(BMKY2013B03-1)
Foundation Item: Scientific Research Project of State Secrets Bureau (BMKY2013B03-1)

的方法仍然占主流地位，目前国内外还没有成熟通用的自动化关键函数定位系统。常用的函数定位方法有以下几种。

1) 设置API函数断点

API（application programming interface）函数是系统为应用软件提供的服务接口。大多数应用软件都是通过API函数访问文件、注册表、网络等资源。因此可以根据逆向的目标，对相应的API函数设置断点，然后以此为分析线索定位关键函数的位置[1]。这种方法对于保护机制简单的软件是有效的，定位效率也很高。但是现在很多商业软件很少甚至不调用这些通用的API函数，而是采用其他方法获取用户输入。这时就需要设置大量的API函数断点来分析所有可能的用户输入，但过多设置API函数断点将会导致逆向效率低下。

2) 设置内存断点

设置内存断点是通过修改断点所在的内存页属性实现的。例如设置访问断点就可设置页的属性为禁止访问。这样当软件访问到这个内存页时就会引发异常，并由调试器捕获[1]。

通过设置内存断点可以实现数据追踪，从而定位处理该数据的关键函数。但该方法效率过于低下，工作量很大。另一方面，当前一些软件已经可以通过检测内存页属性判断是否被调试器安装了内存断点，这些检测方法也使内存断点的设置受到了限制。

3) 利用调试器提供的定位功能

目前一些比较著名的动态调试软件如Ollydbg、SoftIce等都具备了消息调试的功能。利用该功能逆向分析人员可以通过在OllyDbg给出的窗口过程函数设置条件断点对目标消息响应函数进行定位[1]。虽然这种方法可以对关键函数进行较高精度的定位，但只适用于使用事件驱动模式编程的界面程序。

4) 使用密码函数的识别工具

随着人们安全意识的增强，密码函数已是很多软件的重要组成部分，因此出现了一批针对密码函数和编码函数的识别工具，如Krypto Analyzer、Findcryptplugin和SnD Crypto Scanner等。这类工具的工作原理主要是依据大多数的密码函数（如AES、MD5等）都包含了一些特殊的指令序列和常数数据，例如特定的s盒、置换表、初始向量等等。因此这些工具仅适用于已知的具有特征的密码算法，对于运用一般数学计算的密码函数（如RSA）和隐藏了常数特征的密码算法无法进行定位识别。

5) 自动化识别技术

基于人工干预的定位方法，对分析人员的熟练程度和逆向经验要求较高，分析工作量很大，效率很难提高。因此，近年来逐渐出现了关键函数自动化定位的相关理论。2000年，CHEN K R等人提出使用依赖图(dependence graph)[2]的方法进行软件的功能代码定位，但该技术需要分析者对目标软件非常熟悉，否则分析者无法确定逆向分析工作的切入点。2005年，KOSCHKE R等人提出了“形式化分析”(formal concept analysis)[3]的方法，能够有效定位一组给定功能的实现代码。但是这种技术的应用受到需要定位的功能的数量限制，不能适用于较大数量功能集的定位需求。同年，EISENBERG A D等人对动态分析数据使用了“启发式排名”(ranking heuristic)[4]技术，通过将代码元素与给定功能的相关性进行量化并排名实现了特定功能代码的自动化定位。但是针对不同的功能这种量化方法缺乏通用性，并且在排名中采用的启发式算法也有待进一步的评估。2009年谢裕敏等人提出了一种消息响应函数定位方法[5]。该方法利用MFC(Microsoft foundation classes)框架软件的封装特性以及代码数据的特征快速定位MFC软件的消息响应函数地址。2011年，上海交通大学邓超国等人提出一种基于全系统仿真和指令流分析的二进制代码分析方法[6]。该方法以bochs模拟器为分析平台，采用软件切片分析技术对软件二进制代码进行指令分析和数据分析，能够实现对恶意软件核心模块的定位。然而，在指令流记录的过程中产生的海量信息对虚拟系统的执行效率和存储空间提出了限制。

从上述文献的研究现状中不难发现，虽然关键函数定位技术取得了一定的研究成果，但是其中用到的技术手段大多是为解决某一特定问题而设计的，目前还没有一套完整通用的关键函数定位技术。

3 软件执行差异比对的原理

一般地，软件的输入（如键盘输入、鼠标点击和网络数据接收等）发生改变会导致软件的执行过程也发生改变。比如带图形界面的加解密软件，其加解密算法函数会在用户点击加解密按钮后被执行。如果用户在执行过程中没有点击按钮，则该加

解密函数不会出现在软件执行轨迹中。如果将这两次软件执行轨迹进行比对，差异结果中必然含有解密函数。软件执行差异比对的思想是将软件多次运行，并通过改变软件的输入来改变软件的执行轨迹，然后将多次运行得到的执行轨迹分为触发关键函数执行和不触发关键函数执行的 2 个组别，最后通过将这两组执行轨迹进行比对来定位关键函数的位置。

理想情况下，可以通过改变软件输入来精确控制软件流程，并通过差异比对直接定位关键函数。但是对于大多数应用软件，软件的输入很难控制，比如系统环境、运行时间和消息接收等都可能是软件的输入。这时差异结果中会含有与关键函数定位无关的函数，称为“噪声函数”。为此，本文提出用了“1+N 次”比对方法进行差异比对。所谓“1+N 次”比对是这样的一种方法：首先在触发关键函数执行的情况下将软件执行 1 次，然后在不触发关键函数执行的情况下将软件执行 N 次，最后将这 2 组执行轨迹进行比对。将软件在不触发关键函数情况下执行 N 次的目的在于多收集各种由于输入不确定带来的噪声函数，以减小差异比对误差。

4　软件执行轨迹的获取技术

4.1　软件执行轨迹

软件的执行轨迹（execution trace[7]）是软件执行过程的记录信息，可以用指令级、函数调用级和系统调用级等不同粒度的软件执行信息描述执行轨迹。指令级粒度的执行轨迹记录的信息量最完整，但需要对软件进行指令级别的监控，将会严重降低软件的运行速度，而且产生的数据量非常庞大。所以指令级轨迹记录只适合于进行软件的局部代码分析。系统调用级粒度的轨迹记录优点在于不会过多影响软件运行速度，而且产生的数据较少，但记录粒度过粗，会丢失很多信息[8]。函数调用级粒度的执行轨迹记录不会对软件运行速度产生较大影响，而且产生的数据量适中，符合关键函数定位的需要。所以最终选择用函数调用来描述软件的执行轨迹。下文首先给出软件执行轨迹的相关定义。

一次函数调用一般来说包含 2 个基本要素：调用点和函数入口点。在软件执行过程中经常会对同一函数会进行多次调用，从软件执行轨迹上看，每次函数调用都应看作是不同的调用事件。因此为对调用事件进行区分，可按调用事件发生的时间顺序从 1 开始对其进行编号。

定义 1　函数调用过程

从软件执行的第一个函数调用开始，其函数调用过程 S 可以由一个序列表示 $f_1, f_2, \cdots, f_i$，其中每个 f_i 表示第 i 个时刻软件所调用的函数，每个 f_i 可定义为一个二元组$<C_i, E_i>$，其中，C_i 表示函数调用时的调用点地址，E_i 表示函数调用时的入口地址。

定义 2　函数调用集合

函数的调用集合定义为 $V=\{E_i|i=1,2,\cdots,n\}$。

本文使用静态反汇编工具 IDA Pro[9]对程序进行分析，在给某个函数命名时，它会使用该函数的入口地址和一个表示该位置类型的前缀 sub_进行命名,即 sub_xxxxxx 表示地址 xxxxxx 处的子例程。因此在下文中，使用调用函数的名称表示函数调用集合 V。

定义 3　函数调用基本集

函数调用基本集的定义为 $B=\{E_i|i\in N$，且 $i?j, E_i\neq E_j$。

函数调用集合包含一次函数调用过程中所有的函数调用，由于在不同时刻 $i, j(i?j, i, j=1,2$，可能有相同的函数调用，因此可能出现 $E_i=E_j$。函数调用基本集中每个元素对于一次函数调用过程来说是唯一的，即 $E_i\neq E_j$。

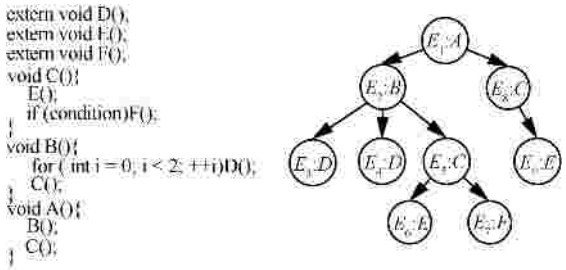
定义 4　动态函数调用树（dynamic call tree）

程序一般具有唯一的入口函数，其他函数均直接或间接地由该入口函数调用，因此程序的一次执行所产生的函数调用可表示为一棵有序树（ordered tree），称作动态函数调用树，简称为函数调用树，记为 $T=\{V,G\}$。其中，V 为函数调用集合，G 为函数调用的二元关系集合且 $G\subset V\times V$。给定 V 中的 2 个元素 E_i、E_j，E_i 为 E_j 的父节点当且仅当在 E_i 是中调用了 E_j；E_i 是 E_j 的左兄弟当且仅当 E_i 先于 E_j 调用且两者拥有相同的父亲。给定元素 E_i，则 E_i 到根节点的距离为 E_i 的深度，记为 $h(E_i)$。

用动态函数调用树表示程序的执行轨迹，是对程序执行过程的有效抽象，可在一定程度上反映程序的执行过程，同时可避免过多地暴露程序的执行细节，减少了执行轨迹记录与分析的数据量。

下面给出了软件执行轨迹的一个例子。如图 1 所示，图 1(a)是软件的代码。A 是主函数，调用了函数 B 和 C；B 中有个循环结构，C 中有个条件判

断。图 1（b）是该软件执行对应的一棵动态函数调用树。



(a) 示例程序 (b) 动态函数调用树

图 1 示例程序及其动态函数调用树

该动态调用树对应的函数调用集合 $V=\{E_1,E_2,E_3,E_4,E_5,E_6,E_7,E_8,E_9\}$，函数调用基本集为 $B=\{A,B,C,D,E,F\}$，函数调用深度 $h(E_1)=0$，$h(E_2)=h(E_8)=1$，$h(E_3)=h(E_4)=h(E_5)=h(E_5)=2$，$h(E_2)=h(E_8)=3$。

4.2 函数调用信息的获取

获取程序执行轨迹首先要获取程序执行过程中的函数调用信息，其中包括函数识别和函数调用关系获取 2 部分内容。为了获取这些信息，本文采用了动态二进制插桩技术。

动态二进制插桩(DBI, dynamic binary instrumentation)技术[10]的研究始于 20 世纪 90 年代。该方法能够在不影响程序执行结果的前提下，根据用户的需求，通过在程序动态执行过程中插入额外的分析代码，达到监视程序执行过程的目的，并且该过程无需程序源码，也不必对源程序重新编译、链接。由于其具备高效、透明和功能接口丰富等特点，近年来在软件自动化逆向分析领域逐渐展露头角。

动态二进制平台是一种利用动态二进制插桩技术对程序进行分析的软件框架。迄今为止，国外已经出现了多个使用 DBI 技术的二进制分析平台，如 DynamoRIO[11]、Valgrind[12]和 Pin[13]等。通过这些平台提供的丰富接口，用户可以方便地获取程序运行期间的堆栈内存、指令操作数等信息，在汇编指令级对程序进行动态监控。本文采用 Pin 作为动态二进制分析平台。

根据调用者的不同，可以将函数调用分为普通函数和回调函数（callback function）。普通函数是应用程序代码主动进行的函数调用。回调函数是应用程序提供给 Windows 操作系统 DLL 或其他 DLL 调用的函数，一般用于截获消息、获取系统信息和处理异步事件等。普通函数的调用点在用户代码空间，回调函数的调用点在系统 DLL 空间。

一般地，程序通过 call 指令进行函数调用、通过 ret 指令进行函数返回。call 指令和 ret 指令通常是成对出现的。但对于恶意软件和一些被混淆的程序，部分 call 和 ret 指令并不是进行函数调用和函数返回，而是被精心构造用于实现普通跳转功能，以达到混淆程序流程的目的。

另外，正常的程序中也会调用_alloca_probe 和_seh_prolog 等栈不平衡函数。这 2 个函数都是通过 call 指令进行调用、ret 指令返回，但函数调用前后函数调用栈是不平衡的，所以进行函数识别时也不能简单使用栈平衡原则进行函数识别。

为了应对上述情况，本文使用了影子栈(shadow stack)技术对函数进行识别。影子栈的设计如图 2 所示。其中，栈元素由函数入口点 ESP 寄存器的值和 call 指令的返回地址 2 部分组成。

图 2 影子栈结构示意

利用影子栈，通过对 3 类指令进行处理实现函数识别。具体方法如下。

1) call 指令

遇到 call 指令时仅记录信息，标记标志，还不能完成函数的识别。处理方法如下：

(a) 根据当前指令 ESP 寄存器的值更新影子栈；

(b) 将 call 指令的目的指令 ESP 寄存器的值（等于当前 call 指令的 ESP 减 4）和 call 下一条指令地址压入影子栈中；

(c) 判断是否进入系统 DLL 空间。如果进入，则将代码空间标志 SpaceFlag 设置为 true。该标志会被用来识别回调函数。

2) ret 指令

遇到 ret 指令时，处理方法如下。

(a) 根据当前指令 ESP 寄存器的值更新影子栈。

(b) 判断影子栈顶的 ESP 值是否等于当前指令 ESP 寄存器的值。如果相等，说明 call、ret 匹配成功，普通函数调用被识别；否则转到 3)。

(c) 判断栈顶返回值是否等于 ret 指令的目的地

址。如果相等，则说明_alloca_probe 等栈不平衡函数被识别；否则，说明该 ret 指令不是函数返回。

(d) 判断是否进入系统 DLL 空间。如果进入，则将代码空间标志 SpaceFlag 设置为 true。

3) trace 头部指令

遇到 trace 头部指令进行如下处理。

(a) 判断代码空间标志 SpaceFlag 是否为 true。如果是，说明从系统 DLL 空间跳转过来，此时将 SpaceFlag 设置为 false，转到(b)；否则不进行处理。

(b) 根据当前指令 ESP 寄存器的值更新影子栈。

(c) 判断影子栈顶记录的返回值是否等于当前指令地址，如果相等说明识别出 API 函数调用返回；否则说明识别出回调函数，转到(d)。

(d) 将当前指令 ESP 寄存器的值和 ESP 寄存器指向的值压入影子栈中。

函数的识别流程如图 3 所示。

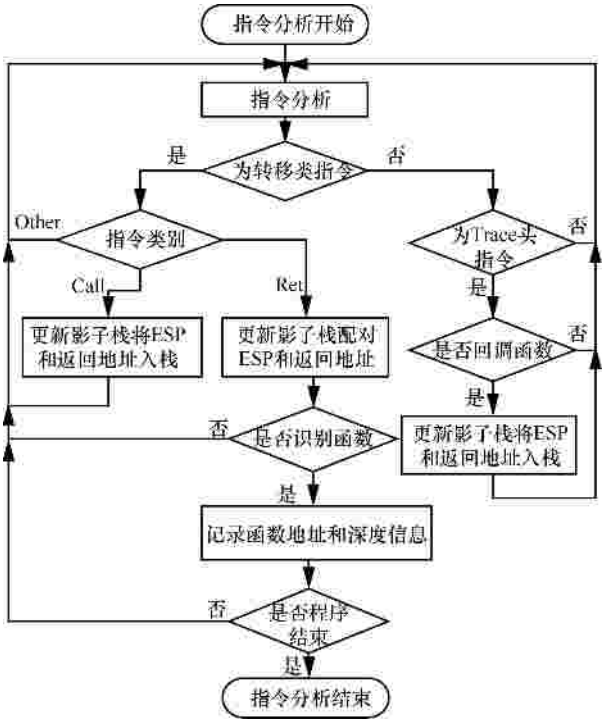


图 3　函数的识别流程

当函数被识别时，取影子栈的深度为函数的调用深度，这样就可以得到一个带深度信息的函数调用集合。对该函数调用序列进一步处理可以生成一颗函数调用树，即程序执行轨迹。

图 4 是对 Winrar3.7 执行轨迹进行动态插桩得到的结果片段，其形式可表示为(*Num*)[*Depth*][+/-]: Address1 Address2 [*Esp*]，其中，*Num* 表示函数调用的序号，*Depth* 表示调用的深度，“ + ”表示此处为函数调用，此时 Address1，Address2 分别表示函数的调用点地址和函数的入口地址，*Esp* 代表栈顶指针的值；“ - ”表示此处为函数返回，此时 Address1，Address2 分别表示函数的结束点地址和函数的返回地址。

```
(1723042) [8] [+]:4211ba 421174 f50d8
(1723043) [9] [+]:42119e 42a2e4 f50d0
(1723044) [10] [+]:42a30b 40c290 f50a8
(1723045) [11] [+]:40c291 40ceb8 f50a0
(1723045) [-] :40cf9d 40c296
(1723044) [-] :40c296 42a310
(1723046) [10] [+]:42a453 42a7a4 f50a8
(1723046) [-] :42a7a4 42a458
```

图 4　Winrar3.7 的执行轨迹片段

以第 1 723 045 次调用为例，函数调用点地址为 0x40c291，入口地址为 0x40ceb8，结束地址为 0x40cf9d，返回地址为 0x40cf96。

```
.text:0040C290  sub_40C290   proc near
.text:0040C290
.text:0040C290               push   edx
.text:0040C291               call   sub_40CEB8
.text:0040C296               retn
.text:0040C296  sub_40C290   endp
```

图 5　AES 函数调用点的反汇编结果

5　基于函数调用集合的差异比对

5.1　函数调用集合差集计算

基于函数调用集合（见定义 2）的差异比对方法是通过对触发关键函数时的函数调用集合和不触发关键函数时的函数调用集合进行差集运算，然后从差集中找到关键函数。

集合差异比对方法的具体步骤如下：

1) 执行软件，输入数据触发关键函数执行，得到函数调用集合 F_0；

2) 执行软件，输入数据但不触发关键函数执行，操作过程尽量与步骤 1）保持一致，以减少噪声函数干扰；

3) 转到 2）直到软件运行 N 次。得到函数调用集合 $F_1, F_2, F_3, \cdots, F_N$；

4) 将不触发关键函数执行的函数调用集合合并。得到集合 $F = \bigcup_{i=1}^{N} F_i$；

5) 取 F_0 与 F 的差集，记为 F_{diff}。$F_{\text{diff}} = F - F_0 = \{x \mid x \in F_0 \ , \ x \notin F\}$。

通过上述步骤，最后得到的差集 F_{diff} 就是基于函数调用集合的差异比对结果。

上述方法有效实施的一个前提是如何给出不同的输入从而触发/不触发关键函数。这个问题根据分析目标的不同有不同的解决方案。如为了分析网络聊天软件的数据通信机制。可在登录后不进行任何操作，记录一组不触发关键函数的数据；登录后发送一条消息，记录过程，这就得到了一个触发关键函数的样本轨迹。

5.2 差异函数集合聚类分析

通过函数调用集合差集计算得到的差集 F_{diff} 仍然存在 2 个问题：其一，F_{diff} 中包含了关键函数，但同时也包含了噪声函数；其二，由于 F_{diff} 中的函数已经去除了函数调用信息，原本有调用关系的函数被分割成孤立的函数调用，不便于关键函数的进一步定位。

为了解决上述问题，本文使用了聚类分析（cluster analysis）的方法对差异函数集合中的元素进行分类。通过聚类分析将数据对象分组成为多个类或簇（cluster），在同一个簇中的对象之间具有较高的相似度，而不同簇中的对象差别比较大[14]。为此，在现有聚类方法的基础上，提出了一种利用函数调用关系的聚类算法。下面先给出与差异集合聚类分析算法相关的定义。

定义 5 差异集合中函数 u 和 v 的距离。假设 a 是函数 u 和函数 v 在函数调用树上深度最大的共同祖先。则 u 和 v 的距离定义如下

$$d(u,v)=h(u)+h(v)-2h(a)$$

其中，h 表示函数的深度（见定义 4）。

定义 6 差异集合中函数 u 和簇 C 的距离定义为

$$d(u,C)=Min(d(u,e)),e\in C$$

差异函数聚类算法的处理流程如下。

1) 从差异集合中选择部分函数作为初始的簇中心。这部分函数需要满足的条件是：函数是差异函数但该函数的父函数不是差异函数。

2) 选择剩余的一个函数，根据其与各个簇的距离，将它赋给与其距离为 1 的簇。函数与簇距离为 1 时表示该函数是簇中某个函数的子函数。

3) 转到 2）直到各个簇的规模不再变化。

4) 将各个簇中的函数组织成为一棵函数调用树。

最终，差异函数经过聚类之后形成一个差异函数调用树森林。由于逆向人员关心的关键函数一般都执行复杂的功能，所以在该森林中表现为规模较大的差异函数调用树。而噪声函数一般表现为规模较小的函数调用树。

6 测试分析

6.1 功能测试

本节选取 WinRAR3.7 进行详细的测试分析。WinRAR 是一款被广泛应用的文件压缩软件，支持多种格式的压缩文件，可使用 AES 算法对压缩数据进行加密。WinRAR 提供了一个命令行版本程序为 Rar.exe，利用基于程序执行轨迹差异比对定位其中的加密函数，结果如下所示。

将 Rar.exe 运行 4 次，其中 1 次执行加密操作，其余 3 次不执行加密操作。测试结果如表 1 所示。

表 1 WinRAR 的函数调用集合比对数据

函数调用统计	触发关键函数执行	不触发关键函数执行		
		第 1 次运行	第 2 次运行	第 3 次运行
函数集合数量	562	550	551	550
函数调用总次数	1 377 257	56 612	56 795	56 676

经过函数调用集合比对后，共得到 10 个差异函数。然后利用函数调用关系将这些差异函数进行聚类，可以分成 5 组。结果如表 2 所示。

表 2 WinRAR 的函数调用集合差异比对结果

聚类组别	函数所在模块	函数地址偏移	函数调用次数
1	Rar.exe	0x2b9c8	1
	Rar.exe	0xce54	1
2	Rar.exe	0xd070	1
	Rar.exe	0xc290	3
3	Rar.exe	0xceb8	3
	Rar.exe	0xd31c	17
4	Rar.exe	0xb5ec	1
	Rar.exe	0x3ff0	1
5	Rar.exe	0x1f24	1
	Rar.exe	0x24e0	1

根据运行结果，可以得出关键组别为第 3 组，其中加密函数地址在 Rar.exe 的 0xceb8 偏移处。手

工调试及第三方加解密程序比对表明，该函数确实为 AES，运行时实际地址为 0x40ceb8，0xc290 偏移处为加密函数调用点地址，0xd31c 偏移处为加密函数内部的子函数调用。

为了验证本文方法的准确性和全面性，本文选取了更多的测试用例来进行验证，具体测试清单如表 3 所示。

表 3　测试用例清单

程序名称	编程框架	是否加壳	程序功能说明	需要定位的关键函数
Notepad	Other	否	Windows 自带的笔记本程序	文件保存函数
HashCalc	MFC	否	哈希值计算	MD5 算法函数
FirePassword	Other	是	Firefox 密码获取程序	密码获取核心函数
dc0decrackme	Delphi	否	注册码验证的 crackme 程序	注册码验证核心函数
RSATool2v1.10	MFC	是	RSA 算法加解密程序	RSA 算法函数

对各测试用例的关键函数进行定位，定位过程中产生的部分数据及关键函数定位结果如表 4 所示。

表 4　测试用例运行数据与关键函数定位结果

程序名称	函数规模	函数调用次数	关键函数地址
Notepad	53	3 686	0x1003D6D
HashCalc	622	135 421	0x41AB13
FirePassword	620	22 354	0x401EC4
dc0decrackme	982	128 726	0x441C08
RSATool2v1.10	154	1 371 024	0x40D130

其中，函数规模是指运行时调用不同函数的个数；函数调用次数是指运行时调用函数的总次数；由于程序加载地址是随机的，关键函数地址指该函数的默认加载地址。

6.2　性能测试

对各测试用例进行关键函数定位的过程中，各个模块的运行时间如表 5 所示。

通过表 5 的数据可以看出，各个子功能的运行时间随着程序的函数规模和函数调用次数增加而增加。分析千万级次函数调用的程序时数据分析模块耗时约 300 s，关键函数定位模块耗时约 60 s，能够满足实际应用的需求。

表 5　各个模块运行时间

程序名称	调用树分析/ms	函数统计/ms	其他/ms	集合比对/ms
WinRAR3.7	30 123	11 582	6 162	642
Notepad	196	69	28	29
HashCalc	10 038	5 887	1 695	532
FirePassword	28 192	12 215	2 941	527
dc0decrackme	4 063	1 632	635	663
RSATool2v1.10	21 102	12 782	3 282	98

7　结束语

本文首先介绍了程序执行差异比对的原理，然后给出了程序执行轨迹的相关定义和程序执行轨迹的获取方法，最后讨论了基于函数调用集合的执行轨迹差异比对方法

实验表明，本文提出的方法针对加密类软件、注册码验证类软件等具有很好地识别效果和通用性。能够正确地获取程序的函数调用信息和执行轨迹信息，并在此基础上能够快速准确地定位关键函数。但是当外部输入或运行环境无法触发关键函数或者所有输入始终会触发关键函数时，无法进行程序轨迹差异比对，因此本文所提出方法将失效。如何在保持动态分析优点的基础上实现软件分析的完整性以及在相似执行轨迹的条件下实现定位，将是本课题需要进一步研究的问题。

参考文献：

[1] 段钢.加密与解密(第三版)[M].北京：电子工业出版社,2008.
DUAN G. Encryption and Decryption(Third Edition)[M]. Beijing: Electronic Industry Press,2008.

[2] CHEN K, RAJLICH V. Case study of feature location usin dependence graph[A]. Proceedings of the 8th International Workshop on Program Comprehension[C].Washington DC, USA, 2000. 241-247.

[3] KOSCHKE R, QUANTEJ. On dynamic feature location[A]. AS '05 Proceedings of the 20th IEEE/ACM International Con erence on Automated Software Engineering[C]. New York, USA, 2005. 86-95.

[4] EISENBERG A D, VOLDER K D. Dynamic feature traces: finding features in unfamiliarcode[A]. Proceedings of the 21st IEEE International Conference on Software Maintenance[C]. Washington DC, USA, 2005. 337-346.

[5] 谢裕敏,舒辉,陈建敏等. MFC 消息响应函数的逆向定位[J].计算机应用, 2009,29(5): 1393-1396.
XIE Y M, SHU H, CHEN J M, *et al*. Location of MFC messages processing functions in converse analysis[J]. Journal f Computer

Applications, 2009, 29(5):1393-1396.

[6] **邓超国，谷大武，李卷孺等. 一种基于全系统仿真和指令流分析的二进制代码分析方法**[J].**计算机应用研究**, 2011, 28(4):1437-1441.
DENG C G, GU D W, LI J R, *et al*. Approach of binary code analysis based on full-system emulation and instruction-flow analysis[J]. Application Research of Computers, 2011, 28(4): 1437-1441.

[7] LIU D, ARCUS A, POSHYVANYK D, *et al*. Feature location via information retrieval based filtering of a single scenario execution trace[A]. Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering[C]. New York, USA, 2007.234-243.

[8] YUAN C, LAO N, WEN J R, *et al*. Automated known problem diagnosis with event traces[A]. EuroSys '06 Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006[C]. New York, USA, 2006. 375-388.

[9] EAGLE C. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler[M]. San Francisco: No Starch Press, 2008.

[10] NETHERCOTE N. Dynamic Binary Analysis and Instrumentation or Building Tools is Easy[D]. Trinity Lane, Cambridge: Un versity of Cambridge, 2004.

[11] BRUENING D L. Efficient, Transparent and Comprehensive Runtime Code Manipulation[D]. Amherst: Massachusetts Institute of Technology, 2004.

[12] NETHERCOTE N, SEWARD J. Valgrind: a framework for heavy-weight dynamic binary instrumentation[J]. ACM SIGPLAN 2007, 42(6):89-100.

[13] LUK C K, COHN R, MUTH R, *et al*. Pin: building customized program analysis tools with dynamic instrumentation[A]. Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI)[C]. New York, USA, 2005.190-200.

[14] HAN J W. Data Mining-Concepts and Techniques[M]. Beijing: China Machine Press, 2006.

作者简介：

康绯（1972-），女，河南周口人，硕士，信息工程大学副教授，主要研究方向为网络信息安全。

王乾（1984-），男，河南民权人，硕士，中国北方电子设备研究所助理工程师，主要研究方向为网络信息安全。

肖亚南（1990-），女，湖南常德人，信息工程大学硕士生，主要研究方向为网络信息安全。

黄荷洁（1989-），男，陕西宝鸡人，信息工程大学硕士生，主要研究方向为网络信息安全。