第 65 卷 第 2 期
2019 年 4 月

武汉大学学报(理学版)
J. Wuhan Univ. (Nat. Sci. Ed.)

Vol. 65 No. 2
Apr. 2019,185~194

DOI: 10.14188/j.1671-8836.2019.02.007

基于内存对象访问序列动态胎记的程序同源性判别方法

陈 铜[1,2],赵 磊[1,2†],王丽娜[1,2],汪 润[1,2]

1. 武汉大学 空天信息安全与可信计算教育部重点实验室,湖北 武汉 430072;
2. 武汉大学 国家网络安全学院,湖北 武汉 430072

收稿日期:2018-08-21 † 通信联系人 E-mail:leizhao@whu.edu.cn
基金项目:国家自然科学基金(61876134,61672394);国家重点研发计划(2016YFB0801100);国家自然科学联合基金重点支持项目(U1536204)
作者简介:陈 铜,男,硕士生,现从事软件安全方面的研究。E-mail:leonids_cu@qq.com

摘 要:针对现有二进制程序同源性判别方法受限于特定编程语言或环境、难以应对复杂的代码混淆攻击、易受依赖库影响等问题,提出了一种基于内存对象访问序列动态胎记(dynamic birthmarks based on memory object access sequences, DBMOAS)的程序同源性判别方法。该方法将程序对数据结构的访问顺序流作为程序语义的一种鲁棒性特征并加以分析,能较好地应对复杂的代码混淆攻击;基于动态污点分析,表征程序的数据结构,解决了二进制程序缺少数据结构与类型的语义表示问题。为验证DBMOAS方法的可信性和弹性,在窗口大小取值不同的情况下,测试具有相似功能的独立程序间的相似度;针对不同编译器、编译选项、混淆方法、版本迭代产生的同源样本,测试程序间的相似度。实验结果表明,本文方法能有效判别程序间的同源性,可信性评估中误判率仅为6.7%,弹性评估中无漏判情况。

关 键 词:软件胎记;内存对象;程序同源性判别

中图分类号:TP 309 **文献标识码**:A **文章编号**:1671-8836(2019)02-0185-10

Software Homology Detection with Dynamic Birthmarks Based on Memory Object Access Sequences

CHEN Tong[1,2], ZHAO Lei[1,2†], WANG Lina[1,2], WANG Run[1,2]

1. Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Wuhan University, Wuhan 430072, Hubei, China;
2. School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, Hubei, China

Abstract: A method called DBMOAS (dynamic birthmarks based on memory object access sequences) for identifying software homology is proposed to solve the problems that the existing methods have limitation on specific programming languages or environments and are difficult to deal with complex code obfuscation attacks and susceptible to dependent libraries. This method regards the access sequence of the data structure in the program as a robust feature of the program semantics and then analyzes it, which does well with complex code obfuscation attacks. A data structure representation method based on dynamic taint analysis is designed to solve the problem of lacking the semantic representation of data structure and type in binary programs. To verify the credibility and resilience of DBMOAS method, the similarity between independent programs with similar functions was tested under different window sizes, and the similarity between programs was also tested for homologous samples generated by different compilers, compilation options, obfuscation methods and version iteration. The experimental results show that the proposed method can effectively identify the homology of the programs and the false positive rate in the credibility test is only 6.7% without false negatives in the resilience test.

Key words: software birthmarks; memory object; software homology detection

引用格式:陈铜,赵磊,王丽娜,等. 基于内存对象访问序列动态胎记的程序同源性判别方法[J]. 武汉大学学报(理学版),2019,**65**(2):185-194. DOI:10.14188/j.1671-8836.2019.02.007.

CHEN Tong, ZHAO Lei, WANG Lina, *et al*. Software Homology Detection with Dynamic Birthmarks Based on Memory Object Access Sequences [J]. *J Wuhan Univ* (*Nat Sci Ed*), 2019, **65**(2): 185-194. DOI: 10.14188/j.1671-8836.2019.02.007(Ch).

0 引言

当前,二进制程序的同源性判别在软件安全、知识产权保护等领域具有诸多应用,例如恶意软件家族分类[1,2]、软件抄袭检测[3]等。随着知识产权意识的兴起以及以恶意软件泛滥为代表的信息安全问题愈发突出,二进制程序的同源性判别问题已经引起了学术界和工业界的广泛关注。

对于二进制程序同源性判别方法,国内外研究者做了大量工作,主要包括基于软件水印[4]和基于软件胎记[5~8]的检测方法。基于软件水印的检测方法需要在软件发布前在程序中植入标识符,且其易受保留程序语义的混淆攻击破坏,用途受限。基于软件胎记的检测方法通过分析可执行文件提取软件的特征,主要分为静态软件胎记检测方法[9,10]和动态软件胎记检测方法[11,12]。静态软件胎记检测方法分析的是软件的词法及结构特性;动态软件胎记检测方法提取的是程序执行过程中的动态行为数据,如系统调用[13~15]、动态关键指令序列[16]、栈行为[17]等。相比于软件的静态特征,动态行为数据只执行程序目标功能分支,能更清晰地刻画程序语义,且时间消耗更小。但目前的动态软件胎记检测方法仍然存在着以下几点不足:1)只能检测特定编程语言,例如,文献[18~21]提出的方法只能对Java字节码进行检测;2)对于复杂的混淆处理或处于苛刻的使用环境时,检测方法表现不够理想,例如,当系统调用数量少或系统调用被混淆替代时,基于系统调用的动态软件胎记方法对程序同源性判别的效果较差;3)易受依赖库的影响,如SODB方法[17]检测同源程序相似度的结果因受程序依赖库迭代的影响而不够理想。

在二进制程序同源性判别过程中,如何对抗代码混淆是一大难点。许多攻击者为躲避检测,常通过控制流混淆、花指令等方式[22]破坏基于程序结构的软件特征,以逃避检测。为了对抗代码混淆,研究者指出软件胎记的特征应建立在能够反应软件功能的软件语义层面上,如何建立语义特征则是这类方法的关键[16]。

而程序的数据结构是程序语义表示的一种重要载体,程序对数据结构的访问流则是一种重要的程序语义特征。独立开发而功能相似的程序对相同的输入数据流有不同的处理逻辑过程,而恶意的攻击者或抄袭者在生成应用时,新应用和原始应用在处理相同的输入数据时,内部的访问数据流应该是相同或者相似的。并且这种逻辑处理与程序在高级语言层面中被访问使用的数据结构息息相关;从高级语言的层面来看,对程序进行的保留语义的混淆处理大部分都不能改变程序本身固有的数据结构。因此,在低级别的指令层面上与这些数据结构相对应的内存对象往往较好地保留了语义特征。

基于此,本文提出了一种基于内存对象访问序列动态胎记(dynamic birthmarks based on memory object access sequences, DBMOAS)的程序同源性判别方法。该方法可直接对二进制程序进行检测,针对二进制程序缺少语义表示这一问题,设计了基于动态污点分析的数据结构逆向分析方法,利用低级别的机器指令逆向推理出表征高级别数据结构的内存对象[23,24]。以内存对象访问上下文作为程序的语义特征集合,能够应对复杂的代码混淆攻击。

1 预备知识

本节将简要介绍动态软件胎记和内存对象。

1.1 动态软件胎记

描述程序固有语义并且可以惟一标识程序的特征集合称为软件胎记。它是解决二进制代码同源性判别问题的有效手段。动态软件胎记的定义如下[17]:

若p^I是输入数据为I时待检测程序p的动态胎记,那么p^I需要满足以下条件:

1)当且仅当p的输入数据为I时,p^I才可在运行过程中获取。

2)若q是p的拷贝,则两者在输入数据相同的情况下,动态胎记一致。

可信性(credibility):若p、q是功能相近但相互独立的程序,检测得到p、q的相似度$\mathrm{sim}\left(p^I, q^I\right) < 1-\theta$($\theta$代表相似度阈值),则这个胎记具有可信性。

弹性(resilience):若q是p通过语义混淆后的一份复制品,检测得到p、q的相似度$\mathrm{sim}\left(p^I, q^I\right) \geqslant \theta$,则该胎记对这种混淆具有弹性。

1.2 内存对象

数据结构是程序算法实现的载体,而对二进制程序直接进行分析只能获取低级别的寄存器和内存操作,无高级的语义表示。因此,为了表征高级层面的数据结构需要用到内存对象。

内存对象是指程序在高级语言层面的数据结构在低级别指令上对应的操作对象。数据结构在二进制层面上的表现形式即为程序在内存中读写的内存对象。内存对象本质为栈或堆等内存空间中与高级语言层面上数据结构相对应的连续字节。

内存对象根据其内存空间是否会在程序生命周期中被重复利用,可以分为静态内存对象和动态内存对象。静态内存对象包括全局变量、静态变量等;而以栈上的局部变量与堆变量为代表的是动态内存对象。

2　DBMOAS方法

DBMOAS方法的总体思想是首先采用在高级语言层面上与输入数据有映射关系的函数内部数据结构及其在函数执行中的访问过程作为程序特征集合。然后,利用动态污点追踪方式,获取程序的精确数据流向,捕获程序在动态执行过程中用来接收和存储仅与外部输入数据相关的关键内存对象,并且提取这些关键内存对象的属性特征及其访问序列,进行软件胎记的语义建模。最后,计算软件胎记的余弦相似度,根据相似度结果判别是否具有同源性。

2.1　实现难点与方法架构

本文方法在实现中存在以下难点:

1) 如何利用污点追踪所得到的指令序列捕获与数据结构相对应的内存对象,并以此进行逆向推理;

2) 如何减少待检测程序之间栈帧结构差异对内存对象识别的影响,例如如何比较函数内联处理后的二进制程序;

3) 如何通过对内存对象的访问过程提取程序语义,并进行相似度计算。

针对上述问题,本文设计了DBMOAS方法,方法架构如图1所示。 DBMOAS方法主要包括以下3个关键步骤:

1) 关键内存对象访问序列生成:利用动态污点追踪可得到与输入域相关的指令序列,以此作为输入,利用栈帧识别、内存对象分析和归一化表示区分等技术,分析得到软件执行过程中仅与输入数据有关的关键内存对象访问序列。以内存对象在栈帧树中的深度差分表示不同程序内的数据结构,减少程序之间栈帧结构差异对栈帧归一化的影响,以此提高语义模型的鲁棒性。

2) 动态软件胎记生成:以关键内存对象访问序列为输入,通过窗口滑动提取程序中对关键内存对象的访问序列特征,得到内存对象访问子序列的频数键值对,以此作为程序的动态胎记。

3) 动态软件胎记相似度计算:以2个待检测程序的软件胎记为输入,通过键值合并重构频率向量,以余弦距离计算待检测程序频率向量的相似度,判断是否独立或同源。

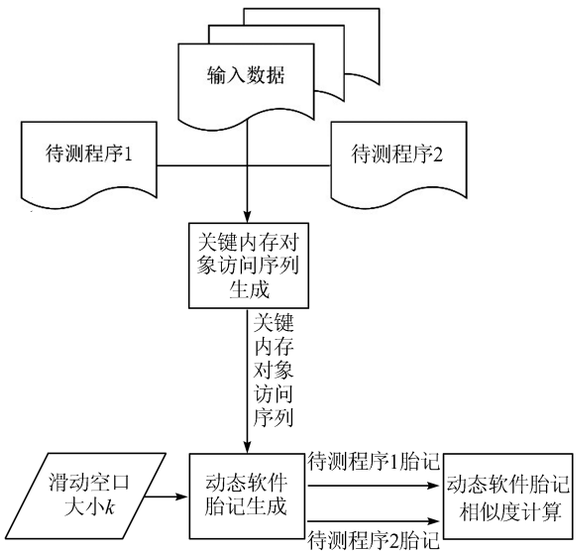

图1　DBMOAS方法架构

Fig. 1　Framework of the DBMOAS method

下文将详细介绍每一个关键步骤。

2.2　关键内存对象访问序列生成

关键内存对象访问序列生成过程如图2所示,主要目的是将动态执行的指令序列转化为结构化数据形式的语义特征,以进行相似度计算。通过栈帧识别和关键内存对象分析手段从污点指令序列轨迹中生成关键内存对象访问序列。

2.2.1　输入数据选取

本文基于Pin污点插桩分析引擎[25]追踪数据流生成轨迹文件,实现DBMOAS方法模型,以输入字段数据缓冲区为初始污点数据。需要注意的是,软件的同源性可能表现在某一功能的代码复用或抄

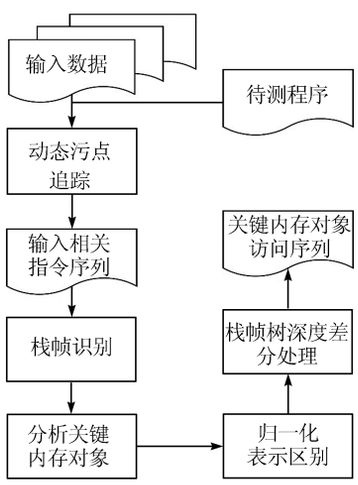

图2　关键内存对象访问序列生成过程

Fig. 2　The generation process of key memory object access sequence

袭,即不同程序之间可能只有部分相应功能的核心代码具有同源性。因此在进行同源性检测时,更加注重不同程序实现相似功能时数据流的相似度,而不只是追求输入数据在动态追踪过程中的整体覆盖率。后者代表的仅是某个功能的指令在单个程序中的比例,而本文是对具有相似功能的多个程序进行同源性判别。为了减少单次输入造成的误差,在下文的实验过程中都以多次重复实验结果的平均值作为最终结果。测试中输入数据的选取原则如下:

1)尽可能覆盖两个程序之间的相似功能,如压缩软件的压缩和解压、解压输出、原始文件删除等。

2)输入数据在测试中实现的功能尽可能一致且输出结果尽可能相似,如压缩软件的输出格式等。

2.2.2 动态污点追踪

动态污点追踪的目的是建立内存对象和数据结构的映射关系,并去除程序中存在的与冗余输入数据无关的数据结构。通过动态污点追踪,获得与输入数据流相关的含有污点标签操作数的指令序列,以此分析生成内存对象的访问序列。

污点追踪过程应遵守如下规则:

1)对不同输入字段所影响的内存字节给予不同的污点标签。

2)在污点传播过程中,若指令的两个源操作数有不同的污点标签,则目的操作数的污点标签为两者的并集。

3)为了丰富对程序语义的描述,除了记录含有污点标签操作数的读行为指令序列,还需要记录目的操作数含污点标签释放污点的写行为指令。

2.2.3 栈帧识别

由于基于变量的切片分析在二进制层面不够准确,同时单纯用基地址标识无法解决指针的别名问题,而在分析内存对象的访问序列时需要区别各个内存对象。因此需要采用栈帧识别,以实现将基地址与栈帧相结合,区分并表示各个内存对象的目的。

栈帧的本质是程序为所执行的函数自动分配的一段逻辑上连续的内存,所以可以根据基地址寄存器的变化和栈帧切换语句对栈帧进行回溯。

常用的栈帧切换语句有push ebp、mov ebp esp、pop ebp、leave等。可以根据对栈帧切换的记录重现栈帧,基本原则就是一个新的栈帧开始于push ebp或mov esp ebp等指令,结束于pop ebp或leave等指令,以此得到程序的栈帧结构,建立栈帧树。

2.2.4 内存对象分析

在指令层面上,为了推导程序与输入数据相关的数据结构,需要分析程序指令中的内存对象。

通过追踪污点数据在内存中的执行访问,可以得到程序在不同栈帧中包含污点标签的内存对象[26]。在某一函数的栈帧中,若某内存地址不是由其他内存地址所派生,则该内存地址可能就是某内存对象的基地址。通过回溯函数栈帧中的执行上下文,分析内存地址的派生关系,以此定位内存空间地址在当前栈帧中的根地址,并将该根地址作为内存对象的基地址。这样就可以识别栈上的局部变量和堆区在栈帧上的指针。

以图3所示代码片段为例,输入字段为文本文件内容,该程序中含有污点且被访问的栈上局部变量为d[1]、d[2]、d[3]、d[4]。在执行内存分析模块后,可以获得程序轨迹文件。图4为对图3所示程序污点追踪的结果,也为对其访问的内存对象的集合。图4中各列的内容说明如表1所示。值得注意的是,在污点追踪的过程中,需判断对内存对象的访问是否为数组间接访问方式。因为普通数组成员的访问可以视作对独立内存对象的访问,数组成员之间的关系不大(如结构体等),但是存在着一些

```
int foo(char *buf)
{
  char d[32];
  strcpy(d,buf);
  if(d[1]!="A")
     a++;
  if(d[2]!="B")
     a++;
  if(d[3]!="C")
     a++;
  if(d[4]!="D")
     a++;
  return true;
}
int main(int ac,char **av)
{
  int fd;
  char *buf;
  if(!(buf=malloc(32)))
     return -1;
  fd = open("./file.txt",O_RDONLY);
  read(fd,buf,32),close(fd);
  foo(buf);
}
```

图3 示例程序代码片段

Fig. 3 Code snippet for the sample program

```
[0] 4235 ebp-43 8 r t1 134513949
[0] 4235 ebp-42 8 r t1 134513976
[0] 4235 ebp-41 8 r t1 134514003
[0] 4235 ebp-40 8 r t1 134514030
```

图4　示例程序污点追踪的部分结果

Fig. 4　Part of the sample program's taint analysis log

表1 污点追踪日志中各列内容

Table 1 Description of each column in the taint tracking log

列号	内容名称	内容说明
1	间接访问标识	判断该内存对象的访问是否为数组间接访问
2	栈帧编号	栈帧在栈帧树中的编号
3	基地址	该内存对象的基地址
4	内存对象大小	该内存对象的大小
5	内存读写标识	r代表从内存中读取、w代表写入内存
6	污点行为标识	t代表被标记上污点、f代表污点被释放,t和f后面的字符代表污点源
7	指令地址	用于判断是否为依赖库

通过数组地址转化后得到的间接访问的成员变量,对这种内存对象需要转化得到其在栈帧中的基地址。

2.2.5　归一化表示区分

在程序函数中一个变量可能会被多次使用,这在二进制层面上来看,代表一个栈帧中的某个内存对象被多次读取或写入。又因为栈是动态分配的,受此影响,不同数据结构在栈帧中的基地址可能是相同的。因此本文通过将内存对象所在栈帧序号与基地址结合,归一化区分同一程序内特定函数中的特定数据结构。

例如图4中的内存对象序列可以归一化表示为六元组序列,分别表示栈帧序号、基地址、大小、内存读写标识、污点行为标识和污点源,结果如下:

$\{\langle 4235,-43,8,0,0,1\rangle,\langle 4235,-42,8,0,0,1\rangle,\langle 4235,-41,8,0,0,1\rangle,\langle 4235,-40,8,0,0,1\rangle\}$

2.2.6　栈帧树深度差分处理

内存对象访问序列经过归一化后可以代表单一程序中的数据流,但这种依赖于栈帧结构的序列不适合直接构造胎记,因为在待测程序功能修改和混淆处理过程中,程序调用的函数链可能会产生变化。比如,因冗余函数或函数内联混淆处理,原程序的某一数据结构所在栈帧的序号可能发生变化。但大部分的内存对象与其在序列中邻近的内存对象同属一个栈帧或其所在栈帧在函数调用栈中的相对偏移并没有变化。

因此,为了减少这种程序之间栈帧结构差异对内存对象识别的影响,可以用内存对象在栈帧树中的相对深度变化而非绝对栈帧序号表示不同程序内的数据结构,所以需要对内存对象所在栈帧在栈帧树中的深度进行差分处理。

利用栈帧切换时栈帧树的深度变化,将内存对象所在栈帧的变化序列与内存对象的访问序列相结合,以构建鲁棒性更强的语义模型。

2.3　软件胎记生成

本文算法首先在栈帧树中查找各个内存对象进行深度差分,再基于k-gram算法[27]对内存对象访问序列进行窗口滑动处理,生成各个序列片段与其访问频数的键值对。设程序的输入数据为I,包含t个输入字段/污点源,算法描述如下。

算法 DBMOAS生成算法

输入: 1) 输入数据为I时,程序p的关键内存对象访问序列D:$\langle d_1,d_2,\cdots,d_n\rangle$

2) 滑动窗口大小:k

3) 栈帧识别时构建的栈帧树:SFTree

输出: 程序p在输入数据为I时的软件胎记

1. for $a=1\rightarrow n$ do:
2. 遍历关键内存对象访问序列,根据内存对象的栈帧编号获取其在栈帧树SFTree中的深度:$D[a]$.depth$\leftarrow$SFTree.getDepth($D[a]$.stackframeid)
3. if $a==1$ then:
4. $D[a]$.depth_diff$\leftarrow 0$
5. else
6. 计算相邻内存对象的深度差值:$D[a]$.depth_diff$\leftarrow D[a]$.depth$-D[a-1]$.
7. End for
8. for $j=1\rightarrow t$ do:
9. 提取出污点源为第j个的内存对象子序列:$D_j\leftarrow$ Extract(D, j)
10. 对序列D_j以窗口大小k滑动,生成长度为k的子序列的集合:$\mathrm{SetD}_j^k\leftarrow$ Slide(D_j,k)
11. $\mathrm{BP}_j^k\leftarrow\{\}$
12. for childseq in SetD_j^k do:
13. 计算每一个子序列childseq在原始序列D_j中出现的频数:BP_j^k[childseq]$\leftarrow$ Freq(childseq, D_j)
14. End for
15. End for
16. $M_p^I\leftarrow[\mathrm{BP}_1^k,\mathrm{BP}_2^k,\cdots,\mathrm{BP}_t^k]$
17. return M_p^I

2.4 软件胎记相似度计算

本文算法通过以下步骤检测程序间的相似度:

1) 胎记初始化:输入数据为I时,通过DBMOAS方法得到待检测程序p的软件胎记为:$\boldsymbol{M}_p^I=[\mathrm{BP}_1^k,\mathrm{BP}_2^k,\cdots,\mathrm{BP}_t^k]$,其中,输入数据$I$由$t$个输入字段构成,$\mathrm{BP}_i^k$代表第$i(i=1,2,\cdots,t)$个输入字段情况下,程序$p$生成的软件胎记。相似地,输入数据为$I$时,待检测程序$q$在相同输入下的软件胎记为$\boldsymbol{M}_q^I=[\mathrm{BQ}_1^k,\mathrm{BQ}_2^k,\cdots,\mathrm{BQ}_t^k]$。

2) 键值合并:令输入字段序号为i时,软件胎记BP_i^k中各个键值对的键集合为$\mathrm{KSet}(\mathrm{BP}_i^k)$,$\mathrm{KSet}(\mathrm{BQ}_i^k)$为$\mathrm{BQ}_i^k$的键集合,此时键值对中的键为关键内存对象子序列,值为子序列的出现次数,分别将各个输入字段情况下待检测程序p和q的软件胎记的键集合相并,如(1)式所示。

$$\mathrm{KS}_i^I=\mathrm{KSet}(\mathrm{BP}_i^k)\cup\mathrm{KSet}(\mathrm{BQ}_i^k) \tag{1}$$

3) 构建频率向量:输入字段序号为i时,分别构建程序p与q以键值集合KS_i^I为新键的频率向量$\boldsymbol{p}_i^k=[p_1,p_2,\cdots,p_m]$与$\boldsymbol{q}_i^k=[q_1,q_2,\cdots,q_m]$,$p_j$如(2)式所示。

$$p_j=\begin{cases}v_r,\ \text{if}\ S_r\in\mathrm{KSet}(\mathrm{BP}_i^k)\\0,\ \text{if}\ S_r\notin\mathrm{KSet}(\mathrm{BP}_i^k)\end{cases},\ S_r\in K_i^I,\ j=1,2,\cdots,m \tag{2}$$

其中,m为键集合K_i^I的大小,v_r表示BP_i^k中以S_r为键所对应的值。

4) 各输入字段余弦相似度比较:输入字段序号为i时,程序q与p的余弦相似度(余弦距离):

$$\mathrm{sim}\left(\boldsymbol{p}_i^k,\boldsymbol{q}_i^k\right)=\frac{\boldsymbol{p}_i^k\cdot\boldsymbol{q}_i^k}{\left|\boldsymbol{p}_i^k\right|\left|\boldsymbol{q}_i^k\right|} \tag{3}$$

5) 带权计算总相似度:在对各个输入字段的内存对象访问序列进行比较后,可以得到对应于各个输入字段的内存对象访问序列相似度,并计算输入数据为I时,待测程序之间的相似度,如(4)式所示。

$$\mathrm{sim}\left(p^I,q^I\right)=\sum_{i=1}^{t}\frac{w_i}{\sum_{j=1}^{t}w_j}\cdot\mathrm{sim}\left(\boldsymbol{p}_i^k,\boldsymbol{q}_i^k\right) \tag{4}$$

其中,p^I和q^I代表输入字段序号为i时,程序p和q的软件胎记,w_i代表第i个输入字段的权重,t为输入字段个数。

6) 相似度判断:计算待检测程序之间相似度后,根据相似度阈值θ,可衡量两程序间的相似度,如(5)式所示。

$$\mathrm{result}=\begin{cases}p\approx q,\ \text{if}\ \mathrm{sim}(p^I,q^I)\geqslant\theta\\p\neq q,\ \text{if}\ \mathrm{sim}(p^I,q^I)<1-\theta\\\text{inconclusive, otherwise}\end{cases} \tag{5}$$

3 实验

为了验证DBMOAS方法的有效性,本文将进行可信性与弹性评估实验,实验结果如表2、表3、图5和图6所示。

表2 不同独立程序间相似度对比

Table 2 Comparisons of the similarity between different independent programs

Name	k								
	2	3	4	5	6	7	8	9	10
bzip2-vs-gzip	0.526	0.196	0.001	0	0	0	0	0	0
bzip2-vs-lrzip	0.568	0.286	0.120	0.051	0.019	0.005	0.003	0.002	0.001
bzip2-vs-pngcrush	0.079	0.043	0.031	0	0	0	0	0	0
bzip2-vs-xz	0.867	0.646	0.365	0.131	0.046	0.008	0.004	0	0
bzip2-vs-md5sum	0	0	0	0	0	0	0	0	0
gzip-vs-lrzip	0.370	0.030	0.003	0.001	0	0	0	0	0
gzip-vs-pngcrush	0.004	0.002	0.001	0	0	0	0	0	0
gzip-vs-xz	0.297	0.059	0	0	0	0	0	0	0
gzip-vs-md5sum	0	0	0	0	0	0	0	0	0
lrzip-vs-pngcrush	0.043	0.010	0	0	0	0	0	0	0
lrzip-vs-xz	0.485	0.276	0.109	0.048	0.024	0.011	0.005	0.002	0
lrzip-vs-md5sum	0	0	0	0	0	0	0	0	0
pngcrush-vs-xz	0.076	0.014	0.011	0.011	0.010	0	0	0	0
pngcrush-vs-md5sum	0	0	0	0	0	0	0	0	0
xz-vs-md5sum	0	0	0	0	0	0	0	0	0
Average Similarity Score	0.221	0.104	0.042	0.016	0.006	0.001	0	0	0

实验过程中,假设测试程序所有输入字段的权重均相同,并且为了减少误差,均采用多次输入数据后构建的软件胎记之间相似度的均值;根据文献[16,28],通常设定相似度阈值为0.7,即相似度大于0.7时,判定为同源程序,相似度小于0.3时判定为独立程序,其他情况下无法判定。

窗口大小k值越大对独立开发程序的误判率越低,对非独立程序的漏报率越高[28]。在本文实验中,将以不同的窗口取值进行可信性评估实验,并选取合适的窗口长度作为最佳窗口长度,在保证软件胎记能识别大多数独立程序的前提下,尽量降低非独立程序的漏报率。

3.1　可信性评估

可信性评估实验中,采用6个功能相近但独立实现的不同程序,测试任意两个程序间的相似度,以验证DBMOAS方法的可信性。根据窗口大小的取值不同一共进行了9轮,共计135组实验,测试结果如表2所示。表2中,"bzip2-vs-gzip"表示bzip2程序与gzip程序在不同窗口大小下的相似度,以此类推;"average similarity score"表示在不同窗口大小取值情况下的平均相似度。

由表2可知,当k为4时,测试程序间的平均相似度仅为0.042,此时DBMOAS方法能较为准确地区分独立程序;当k大于4时,程序间相似度变化较小。当k为4时,虽然有一组独立程序间相似度大于0.3,但更大的k值会使独立程序相似度检测结果更低的同时造成同源程序的漏报率上升。由于本文方法更加注重在容忍一定的独立程序误判率情况下,保证针对同源程序的相似度检测率,而k=4时,DBMOAS方法的误判率仅为6.7%,且无漏判情况。因此在接下来的实验中,设置窗口值为4。

3.2　弹性评估

弹性评估实验中,将分别针对不同编译优化方法、混淆方法、版本迭代产生的同源样本进行实验,验证DBMOAS在各种情况下的检测能力。

3.2.1　不同编译器及编译选项处理后程序的相似度检测

本组实验将对不同编译器及编译选项组合情况下,同一程序的各种版本进行检测。测试对象是pigz程序经过clang(LLVM 3.2)和gcc(GCC 4.6.3)这2种编译器(compiler)用3种编译优化等级(OPT-Level:o1,o2,o3)分别组合编译后,得到的12种版本程序(含调试(debug)版本和发布(release)版本)。

如表3所示,在对12种版本测试程序进行两两组合之后,共有66组不同实验的结果,其中相似度最小值为0.981,最大值为1.000,平均值为0.993,无漏判情况;而相同实验条件下,动态软件胎记SODB[17]方法的相似度平均值只有0.978。实验结果表明,DBMOAS方法在针对不同编译器及编译选项编译后程序的相似度检测方面具有较好的弹性。

表3　不同编译选项相似度检测结果

Table 3　Similarity test results for different compilation options

Compiler	Versions	OPT-Level	LLVM 3.2						GCC 4.6.3					
			Debug			Release			Debug			Release		
			o1	o2	o3	o1	o2	o3	o1	o2	o3	o1	o2	o3
LLVM 3.2	Debug	o1	1.000	0.984	0.984	1.000	0.999	0.999	0.994	0.998	0.999	0.994	0.999	0.999
		o2		1.000	1.000	0.984	0.985	0.985	0.981	0.986	0.985	0.981	0.986	0.985
		o3			1.000	0.984	0.985	0.985	0.981	0.986	0.985	0.981	0.986	0.985
	Release	o1				1.000	0.999	0.999	0.994	0.998	0.999	0.994	0.999	0.999
		o2					1.000	1.000	0.992	0.997	1.000	0.992	0.999	1.000
		o3						1.000	0.992	0.997	1.000	0.992	0.999	1.000
GCC 4.6.3	Debug	o1							1.000	0.995	0.994	1.000	0.994	0.994
		o2								1.000	0.998	0.995	0.999	0.998
		o3									1.000	0.994	1.000	1.000
	Release	o1										1.000	0.994	0.994
		o2											1.000	1.000
		o3												1.000

3.2.2 语义保留混淆转换的相似度检测

在抗代码混淆实验中,首先对JLex程序用Java字节码混淆转换工具SandMark生成30种混淆版本,然后用GCJ打包得到x86可执行文件,将此可执行文件作为测试对象。如图5所示,在JLex的30种混淆版本与原版本的相似度比较中,大多数对栈帧结构无影响或影响较小的混淆方法与原版本的相似度远高于阈值。在对内存对象访问序列用栈帧深度差分处理后,如函数内联等对栈帧结构影响较大的混淆版本与原版本的相似度都在阈值之上。实验结果表明,本文提出的方法在针对语义保留混淆转换的检测上有较好的弹性,平均检测相似度为0.967,无漏判情况。

3.2.3 软件相邻版本的相似度检测

在针对软件相邻版本的相似度检测中,将同一软件相邻发布的不同版本视作仅修改了部分算法的同源软件,测试对象是pngcrush软件的14个相邻版本,发布时间为2009~2013年。通过将DBMOAS方法与SODB方法[17]计算得到的程序间相似度进行对比,得到13组实验结果,如图6所示。

由图6可知,采用本文方法检测得到pngcrush软件1.6.20版本与1.7.10版本的相似度稍低,为0.854。这是因为这两个程序的核心算法有所更改。而SODB方法方面,则是因测试对象依赖库的升级导致SODB方法检测得到的1.7.40版本与1.7.50版本的相似度大幅降低。

实验结果表明,本文方法对判别软件相邻发布版本的相似性具有较高的弹性,能有效判别相邻版本软件之间的同源性,没有漏判情况。与SODB方法相比,DBMOAS方法不受相关依赖库的影响,能更准确地识别软件核心算法的迭代。

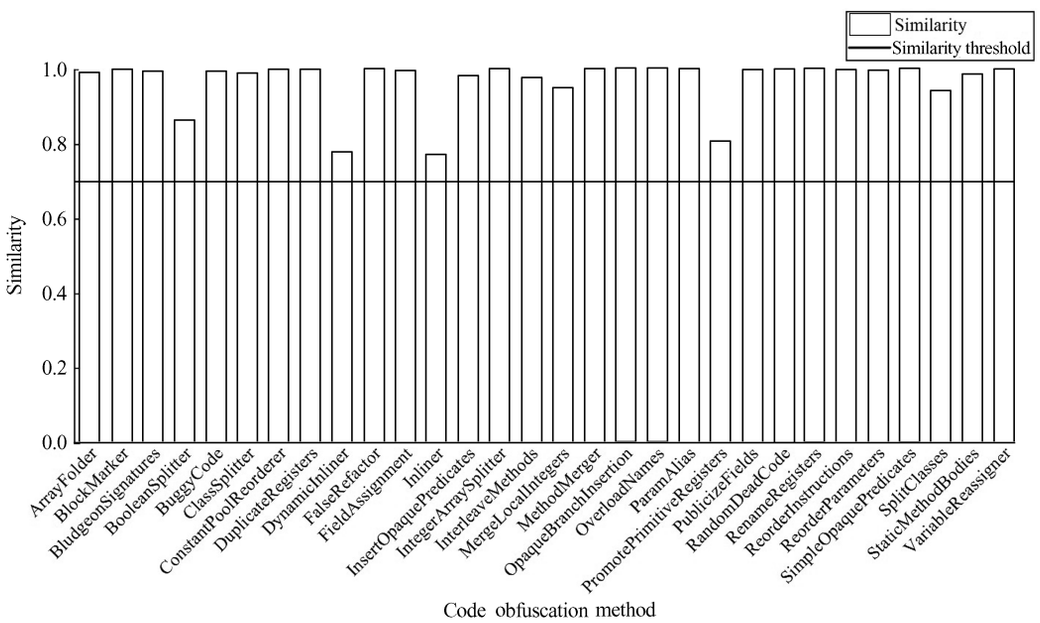

图5 JLex30种混淆版本与原版本相似度比较

Fig. 5 Similarity between JLex's 30 obfuscated versions and the original version

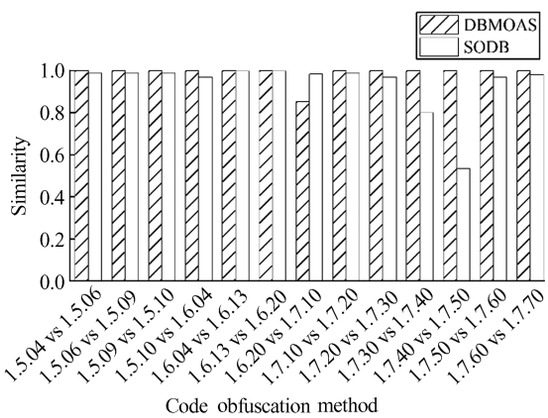

图6 相邻版本相似度

Fig. 6 Similarity between the adjacent versions

4 结束语

由于恶意程序的与日俱增和保护软件著作权的需求愈发迫切,本文针对二进制程序的同源性检测问题,提出了一种DBMOAS程序同源性判别方法。本文贡献如下:

1) 通过研究二进制程序数据结构与内存对象的映射关系,以与输入数据有关的内存对象及其访问序列作为程序特征集合进行相似度度量,提出了一种判别程序同源性的DBMOAS方法。

2) 利用高级语言层面的数据结构逆向推理出

关键内存对象,摆脱了传统程序同源性判别方法中所存在的语言依赖、使用限制等局限。

3) 通过实验验证了DBMOAS方法对不同编译优化方法、混淆方法、版本迭代产生的同源样本具有较高的相似性识别度,且无漏判现象;能够有效识别功能相似但独立开发的程序,误判率仅为6.7%。

在未来工作中,将进行二进制程序基本代码块的相似性研究,从基本代码块层面探究程序具体的修改地址,可用于漏洞更新的逆向分析等,同时将研究如何减少栈帧结构混淆方法对相似度计算的影响。

参考文献:

[1] 肖云倡,苏海峰,钱雨村,等.一种基于行为的Android恶意软件家族聚类方法[J].武汉大学学报(理学版),2016,**62**(5):429-436. DOI:10.14188/j.1671-8836.2016.05.005.
XIAO Y C, SU H F, QIAN Y C, *et al*. A behavior-based family clustering method for Android malwares [J]. *Journal of Wuhan University* (*Natural Science Edition*), 2016, **62**(5):429-436.DOI:10.14188/j.1671-8836.2016.05.005 (Ch).

[2] LI M H, WANG W, WANG P, *et al*. LibD: Scalable and precise third-party library detection in android markets [C]// *IEEE/ACM International Conference on Software Engineering*. Washington D C: IEEE Press, 2017: 335-346. DOI: 10.1109/ICSE.2017.38.

[3] 田振洲,刘烃,郑庆华,等.软件抄袭检测研究综述[J].信息安全学报,2016,**1**(3):52-76. DOI:10.19363/j.cnki.cn10-1380/tn.2016.03.005.
TIAN Z Z, LIU T, ZHENG Q H, *et al*. Software plagiarism detection: A survey [J]. *Journal of Cyber Security*, 2016, **1**(3): 52-76. DOI: 10.19363/j.cnki.cn10-1380/tn.2016.03.005 (Ch).

[4] HAMILTON J, DANICIC S. An evaluation of the resilience of static Java bytecode watermarks against distortive attacks [J]. *IAENG International Journal of Computer Science*, 2011, **38**(1):1-15.

[5] LUO L N, MING J, WU D H, *et al*. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection [C]// *ACM Sigsoft International Symposium on Foundations of Software Engineering*. New York: ACM Press, 2014: 389-400. DOI: 10.1145/2635868.2635900.

[6] SHI G Y. Static software birthmark based on multiple attributes [C]// *International Conference on Mechanical, Electronic, Control and Automation Engineering*. Pairs: Atlantis Press, 2017:403-407. DOI: 10.2991/mecae-17.2017.76.

[7] NAZIR S, SHAHZAD S, RIZA L S. Birthmark-based software classification using rough sets [J]. *Arabian Journal for Science & Engineering*, 2017:**42**(2):859-871. DOI: 10.1007/s13369-016-2371-4.

[8] CHAN P P F, HUI L C K, YIU S M. Heap graph based software theft detection [J]. *IEEE Transactions on Information Forensics & Security*, 2013, **8**(1): 101-110. DOI: 10.1109/tifs.2012.2223685.

[9] KIM D, CHO S J, HAN S, *et al*. Open source software detection using function-level static software birthmark [J]. *Journal of Internet Services and Information Security*, 2014, **4**(4): 25-37.

[10] LIM H I, PARK H, CHOI S, *et al*. A static java birthmark based on control flow edges [C]// 2009 33rd *Annual IEEE International, Computer Software and Applications Conference*. Washington D C: IEEE Press, 2009:413-420. DOI: 10.1109/COMPSAC.2009.62.

[11] FUKUDA K, TAMADA H. A dynamic birthmark from analyzing operand stack runtime behavior to detect copied software [C]// 2013 14th *ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. Washington D C: IEEE Press, 2013:505-510. DOI: 10.1109/SNPD.2013.11.

[12] CHAN P P F, HUI L C K, YIU S M. *Dynamic Software Birthmark for Java Based on Heap Memory Analysis* [M]. Berlin: Springer, 2011. DOI: 10.1007/978-3-642-24712-5_8.

[13] WANG X R, JHI Y C, ZHU S C, *et al*. Detecting software theft via system call based birthmarks [C]//2009 *Annual Computer Security Applications Conference*. Washington D C: IEEE Press, 2009: 149-158. DOI: 10.1109/ACSAC.2009.24.

[14] KO C H, PARK D S, HONG J. An efficient similarity measurement technique using dynamic birthmark based on API [J]. *Information Japan*, 2016, **19**(11):5235-5244.

[15] KIM D, GOKHALE A, GANAPATHY V, *et al*. Detecting plagiarized mobile apps using API birthmarks [J]. *Automated Software Engineering*, 2016, **23**(4):591-618. DOI: 10.1007/s10515-015-0182-6.

[16] TIAN Z Z, ZHENG Q H, LIU T, *et al*. DKISB: Dynamic key instruction sequence birthmark for software plagiarism detection [C]//2013 *IEEE International Conference on High Performance Computing and Communications* & 2013 *IEEE International Conference on Embedded and Ubiquitous Computing*. Washington D C: IEEE Press, 2013:619-627. DOI: 10.1109/HPCC.and.EUC.2013.93.

[17] 范铭,刘烃,郑庆华,等.基于栈行为动态胎记的软件抄袭检测方法[J].山东大学学报(理学版), 2014, **49**(9):9-16.

DOI: 10.6040/j.issn.1671-9352.2.2014.123.

FAN M, LIU T, ZHENG Q H, *et al*. SODB: A novel method for software plagiarism detection based on stack operation dynamic birthmark [J]. *Journal of Shandong University* (*Natural Science*), 2014, **49**(9): 9-16. DOI: 10.6040/j.issn.1671-9352.2.2014.123 (Ch).

[18] PARK H, CHOI S, LIM H I, *et al*. Detecting code theft via a static instruction trace birthmark for Java methods [C]// *IEEE International Conference on Industrial Informatics*. Washington D C: IEEE Press, 2008: 551-556. DOI: 10.1109/INDIN.2008.4618162.

[19] LIM H I, HAN T. Analyzing stack flows to compare java programs [J]. *IEICE Transactions on Information & Systems*, 2012, **95**(2): 565-576. DOI: 10.1587/transinf.E95.D.565.

[20] SCHULER D, DALLMEIER V, LINDIG C. A dynamic birthmark for Java [C]//*Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York: ACM Press, 2007: 274-283. DOI: 10.1145/1321631.1321672.

[21] PARK H, CHOI S, LIM H I, *et al*. Detecting java theft based on static api trace birthmark [C]//*International Workshop on Security*. Berlin: Springer, 2008: 121-135. DOI: 10.1007/978-3-540-89598-5_8.

[22] 赵玉洁,汤战勇,王妮,等.代码混淆算法有效性评估[J].软件学报,2012,**23**(3):700-711. DOI:10.3724/SP.J.1001.2012.03994.

ZHAO Y J, TANG Z Y, WANG N, *et al*. Evaluation of code obfuscating transformation [J]. *Journal of Software*, 2012, **23**(3): 700-711. DOI: 10.3724/SP.J.1001.2012.03994 (Ch).

[23] WANG R, LIU P, ZHAO L, *et al*. deExploit: Identifying misuses of input data to diagnose memory-corruption exploits at the binary level [J]. *Journal of Systems & Software*, 2017, **124**: 153-168. DOI: 10.1016/j.jss.2016.11.026.

[24] LIU Z Y, CRISWELL J. Flexible and efficient memory object metadata [J]. *ACM Sigplan Notices*, 2017, **52**(9):36-46. DOI: 10.1145/3092255.3092268.

[25] LUK C K, COHN R, MUTH R, *et al*. Pin: Building customized program analysis tools with dynamic instrumentation [J]//*ACM Sigplan Notices*, 2005, **40**(6): 190-200. DOI: 10.1145/1064978.1065034.

[26] 任翔宇,谈诚,赵磊,等.识别数据结构的协议格式逆向推理方法[J].武汉大学学报(工学版),2015,**48**(2):269-273,288. DOI:10.14188/j.1671-8844.2015-02-025.

REN X Y, TAN C, ZHAO L, *et al*. Reverse engineering of protocol format via identifying program data structures [J]. *Engineering Journal of Wuhan University*, 2015, **48**(2): 269-273, 288. DOI: 10.14188/j.1671-8844.2015-02-025 (Ch).

[27] MYLES G, COLLBERG C. *k*-gram based software birthmarks [C]//*Proceedings of the* 2005 *ACM Symposium on Applied Computing*. New York: ACM Press, 2005: 314-318. DOI: 10.1145/1066677.1066753.

[28] TIAN Z Z, ZHENG Q H, LIU T, *et al*. Software plagiarism detection with birthmarks based on dynamic key instruction sequences [J]. *IEEE Transactions on Software Engineering*, 2015, **41**(12):1217-1235. DOI: 10.1109/TSE.2015.2454508.

□