(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 113158147 A
(43)申请公布日 2021.07.23

(21)申请号 202110311243.7

(22)申请日 2021.03.24

(71)申请人 中国人民解放军战略支援部队信息工程大学
地址 450000 河南省郑州市高新区科学大道62号

(72)发明人 舒辉 熊小兵 于璞 康绯 杨巨 赵耘田

(74)专利代理机构 郑州明华专利代理事务所(普通合伙) 41162
代理人 高丽华

(51)Int.Cl.
G06F 21/14(2013.01)
G06F 8/74(2018.01)
G06F 8/41(2018.01)

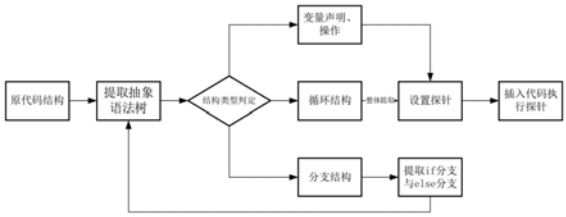
权利要求书2页 说明书7页 附图3页

(54)发明名称

一种基于母体融合的代码混淆方法

(57)摘要

本发明属于网络安全技术领域,具体涉及一种基于母体融合的代码混淆方法。本发明通过将母体代码程序与要保护的代码在结构上进行合并,并通过变量共用等方式使母体代码与要保护的代码之间产生逻辑关联,从而将两个代码合并为不易分割,具有原有代码功能,并对原有执行流图结构进行隐藏的新代码,以此保护原程序。该方法摒弃了传统的混淆方法中对源码形式的过度处理,能够有效解决传统混淆方法难以对抗自动化分析手段、具有明显混淆特征的问题,混淆效果较好;而且该混淆方法适用于大多数编程语言,具有良好的普适性。



CN 113158147 A

1.一种基于母体融合的代码混淆方法,其特征在于:包括以下步骤:

步骤一、选取无害但功能复杂的程序作为母体程序,所述母体程序需要与目标代码属于同种编程语言、具有相同语言规则且不会与目标代码产生冲突;

步骤二、采用抽象语法树结构对母体程序进行遍历,以语法树各结点类型和关系为基础抽象出母体程序的控制流结构,提取出母体程序的抽象控制流图,并对母体程序控制流的关键节点或必经节点进行记录;

步骤三、根据执行流所得到的运行点记录数量对目标代码进行分片,根据获取的母体程序的控制流图和节点信息,选取可融合的代码位置,然后将目标代码分片按照母体程序执行点记录位置顺序与母体程序对应位置进行融合,形成新的控制流图进而形成新的程序。

2.根据权利要求1所述的基于母体融合的代码混淆方法,其特征在于:步骤二具体方法为:对顶层语法树结点的子结点进行遍历,获取母体程序的控制流图,按照遍历顺序对每一个语法树结构类型进行判别,根据结构类型确定插入执行探针的结点位置;

若结构类型为一般变量操作,设置执行探针,直接插入执行探针;

若为特殊语法结构类型,需要根据语法结构的各种操作类型结构进行整合提取,整合提取分为:

(1)若为条件分支结构,设置执行探针,在每一条分支结构内部的开始位置插入执行探针以获取执行当前分支所需要的触发条件;

(2)若为函数调用结构,对函数内部语法结构进行分析,将函数作为一个整体,在整体后插入执行探针以获取函数的参数个数、参数类型以及返回值类型;

根据权利要求2所述的基于母体融合的代码混淆方法,其特征在于:若函数调用结构为递归调用函数,将其视为循环结构,不进行执行探针插入。

3.根据权利要求2所述的基于母体融合的代码混淆方法,其特征在于:若函数调用结构为内部多层次调用函数,则设置深度阈值,并采用深度优先算法,以避免程序陷入多重调用并导致融合失败。

4.根据权利要求1所述的基于母体融合的代码混淆方法,其特征在于:步骤三中对目标代码按照语法树进行完整分片,分片数量不大于运行点记录数量。

5.根据权利要求1所述的基于母体融合的代码混淆方法,其特征在于:步骤三中目标代码分片的融合点优先选择和分片结构类型相同或相似的结构进行融合。

6.根据权利要求1所述的基于母体融合的代码混淆方法,其特征在于:步骤三中目标代码分片与母体程序对应位置融合按照目标代码分片中不同的代码结构采用不同的融合方式,分别为:

(1)对于目标代码分片中的简单语句操作,直接将目标代码分片以原形式填充到母体程序的对应融合位置,所述简单语句操作包括普通表达式和循环体;

(2)将部分代码进行控制流扁平化处理,更改代码结构但不更改程序语义,然后将更改后的代码结构填充到母体程序的对应融合位置;

(3)对于目标代码中存在的分支结构,将母体的分支条件与目标代码分片的分支条件取并集形成新的分支条件,在分支内容上则通过直接以原形式填充的方式,融合到母体代码的对应融合位置。

7.将分片的其他部分插入到条件动作中，设置变量共用，将目标代码分片构建为不透明谓词，并根据记录将母体程序中的变量作为不透明谓词的一个元素构建出与原表达式具有相同语义的融合表达式。

一种基于母体融合的代码混淆方法

技术领域

[0001] 本发明属于网络安全技术领域,具体涉及一种基于母体融合的代码混淆方法。

背景技术

[0002] 逆向分析技术是恶意代码编写者通用的一项技术。通过对现有程序进行逆向分析,挖掘程序内包含的算法逻辑和关键数据,可有效实施软件盗版甚至基于漏洞的黑客攻击。根据商业软件联盟(the software alliance,简称BSA)2018年发布的软件调查显示,全球范围内有高达36%的已安装软件未经合法授权,对软件产业和用户信息安全均构成极大威胁。

[0003] 代码混淆技术是进行软件保护的通用技术,通过将代码在语义、逻辑或表现形式上的变化,对代码原有的执行逻辑进行混淆,达到隐藏代码核心功能的目的。代码混淆主要应用于高级语言,即源码层面的代码混淆,通过对源码形态的变换,生成具有抗逆向功能的可执行程序或脚本程序。主要的源码混淆方式有添加冗余代码,数据加密等。其中添加冗余代码属于控制流混淆的一种,是指将基本块结构相似,数据随机生成的无意义代码片段或不透明谓词插入到原代码控制流中,增加原程序的逻辑分支复杂度。数据加密则是通过将关键内容通过加密的方式,以数据形式进行存储,在使用时再进行解密,例如将源代码以数据形式进行加密,在运行时先解密出源码再执行。

[0004] 然而,上述混淆技术难以对抗以符号执行、模拟执行为主的动态逆向分析技术。研究表明,动态符号执行被广泛应用于添加不透明谓词等通过添加冗余代码的混淆算法的消除:通过分析执行流中每个基本块,对具有分支的基本块进行路径可达性分析,去除不可达路径。而动态模拟执行可将加密数据部分通过模拟解密过程的方式进行提取,以获取加密前形态。动态分析技术有效的消除了传统代码混淆技术对程序破译的阻碍,使得软件保护产业难以取得进一步的突破。

[0005] 显然,传统的代码混淆算法已经难以适应软件保护的需求,混淆方法的创新显得尤为重要,因此,提出一种混淆强度大,方式新颖的代码保护方法是当前的首要解决的问题。

发明内容

[0006] 针对传统的代码混淆算法主要通过增添代码冗余,提高程序复杂度,对抗逆向分析,其混淆强度依赖于代码冗余程度,难以应对日趋高效的自动化分析技术,并且传统的源码混淆方式会产生明显的混淆特征,并暴露混淆手法,使程序更容易越过混淆进行分析的缺陷和问题,本发明提供一种混淆强度大、方式新颖的基于母体融合的代码混淆方法。

[0007] 本发明解决其技术问题所采用的方案是:一种基于母体融合的代码混淆方法,包括以下步骤:

[0008] 步骤一、选取无害但功能复杂的程序作为母体程序,所述母体程序需要与目标代码属于同种编程语言、具有相同语言规则且不会与目标代码产生冲突;

[0009] 步骤二、采用抽象语法树结构对母体程序进行遍历，以语法树各结点类型和关系为基础抽象出母体程序控制流结构，提取出母体程序的抽象控制流图，并对母体程序控制流的关键节点或必经节点进行记录；

[0010] 步骤三、根据执行流刻画所得到的运行点记录数量对目标代码进行分片，根据获取的母体程序的控制流图和节点信息，选取可融合的代码位置，然后将目标代码分片按照母体程序执行点记录位置顺序与母体程序对应位置进行融合，形成新的控制流图进而形成新的程序。

[0011] 上述的基于母体融合的代码混淆方法，步骤二具体方法为：对顶层语法树结点的子结点进行遍历，获取母体程序的控制流图，按照遍历顺序对每一个语法树结构类型进行判别，根据结构类型确定插入执行探针的结点位置；

[0012] 若结构类型为一般变量操作，设置执行探针，直接插入执行探针；

[0013] 若为特殊语法结构类型，需要根据语法结构的各种操作类型结构进行整合提取，整合提取分为：

[0014] (1)若为条件分支结构，设置执行探针，在每一条分支结构内部的开始位置插入执行探针以获取执行当前分支所需要的触发条件；

[0015] (2)若为函数调用结构，对函数内部语法结构进行分析，将函数作为一个整体，在整体后插入执行探针以获取函数的参数个数、参数类型以及返回值类型；

[0016] 上述的基于母体融合的代码混淆方法，若函数调用结构为递归调用函数，将其视为循环结构，不进行执行探针插入。

[0017] 上述的基于母体融合的代码混淆方法，若函数调用结构为内部多层次调用函数，则设置深度阈值，并采用深度优先算法，以避免程序陷入多重调用导致融合失败。

[0018] 上述的基于母体融合的代码混淆方法，步骤三中对目标代码按照语法树进行完整分片，分片数量不大于运行点记录数量。

[0019] 上述的基于母体融合的代码混淆方法，步骤三中目标代码分片的融合点优先选择和分片结构类型相同或相似的结构进行融合。

[0020] 上述的基于母体融合的代码混淆方法，步骤三中目标代码分片与母体程序对应位置融合按照目标代码分片中不同的代码结构采用不同的融合方式，分别为：

[0021] (1)对于目标代码分片中的简单语句操作，直接将目标代码分片以原形式填充到母体程序的对应融合位置，所述简单语句操作包括普通表达式和循环体；

[0022] (2)将循环体代码进行控制流扁平化处理，更改代码结构但不更改程序语义，然后将更改后的代码结构填充到母体程序的对应融合位置；

[0023] (3)对于目标代码中存在的分支结构，将母体的分支条件与目标代码分片的分支条件取并集形成新的分支条件，在分支内容上则通过直接以原形式填充的方式，融合到母体代码的对应融合位置。将分片的其他部分插入到条件动作中，设置变量共用，将目标代码分片构建为不透明谓词，并根据记录将母体程序中的变量作为不透明谓词的一个元素构建出与原表达式具有相同语义的融合表达式。

[0024] 本发明的有益效果：本发明通过代码间结构共用与代码控制流融合的方式，将两个同语言编写的程序代码组成一个新的、同时具有两者完整功能的新代码程序，丰富原有的代码执行流结构，隐藏敏感程序内容，增强程序的抗分析能力与抗同源性检测能力，解决

传统混淆方法混淆强度低的缺点。

[0025] 本发明通过使要保护的代码与无害的母体代码进行融合，摒弃传统的混淆方法中对源码形式的过度处理，通过将核心代码隐藏在复杂的母体代码中，分段隐藏具体的代码行为。在功能完整性上，核心代码通过将自身的控制流结构与母体代码的控制流结构相融合，并选取母体执行流中的关键点或必经点作为融合点，保证了程序的执行正确性与不变性。在静态分析层面，原程序控制流图被破坏，各关键控制流节点加入了新的控制流过程，但主体依然是无害母体的控制流。在动态分析层面，生成的新代码程序在很大程度上依然主要为母体代码的行为，在分析过程中，核心操作分片隐藏在母体代码中，通过变量绑定结构共用的方式融合的代码使母体与核心代码之间存在不易分割的逻辑关系，无法区分母体代码程序与核心代码程序，无害的母体代码为核心代码提供了良好的隐藏效果。

附图说明

[0026] 图1为本发明执行流跟踪流程图。

[0027] 图2为目标代码控制流图。

[0028] 图3为母体代码控制流图。

[0029] 图4为融合后目标代码控制流图。

具体实施方式

[0030] 针对传统的代码混淆算法已经难以适应软件保护的需求的缺陷和问题，本发明提供一种混淆强度大、方式新颖的基于母体融合的代码混淆方法，通过代码间结构共用与代码控制流融合的方式，将两个同语言编写的程序代码组成一个新的、同时具有两者完整功能的新代码程序，丰富原有的代码执行流结构，隐藏敏感程序内容，增强程序的抗分析能力与抗同源性检测能力，解决传统混淆方法混淆强度低的缺点。下面结合附图和实施例对本发明进一步说明。

[0031] 本发明对代码的混淆主要通过母体代码程序与要保护的代码在结构上进行合并，并通过变量共用等方式使母体与要保护的代码之间产生逻辑关联，从而将两个代码合并为不易分割，具有原有代码功能，并对原有执行流图结构进行隐藏的新代码，以此保护原程序。

[0032] 实施例1：本实施例提供一种基于母体融合的代码混淆方法，该方法需要通过结构共用在目标代码和母体程序之间进行代码控制流融合，首先需要选择合适的母体程序作为融合基础，其中母体程序选择无害但功能复杂的程序作为母体代码，母体代码需要满足以下条件：

[0033] (1)与需要保护的代码程序属于同种编程语言，具有相同的语言规则；(2)避免代码间冲突，代码在运行过程中不会与要保护的代码之间产生冲突，例如操作行为冲突等。

[0034] 从算法上遍历控制流图中所有节点，并对必经节点所对应的代码块进行记录，将该部分代码块按照执行时顺序作为融合点进行代码融合。

[0035] 在确定母体代码之后，要对母体代码进行执行流跟踪。在源码层面对母体进行执行流跟踪时，需要利用两个关键点，一是控制流图，另一个是抽象语法树。本发明通过抽象语法树结构进行遍历，以语法树各节点类型及关系为基础抽象出代码控制流结构，以静态

语法分析的方式提取出程序的抽象控制流图，流程如图1所示。具体如下。

[0036] 设置语法树顶层节点和深度阈值，对顶层语法树结点的子结点进行遍历，

[0037] 1、若没有子结点或超过深度阈值，则停止；

[0038] 2、若存在子结点，则对结点结构类型进行判别，并根据节点结构进行对应的执行探针插入操作；如若遇到特殊的代码结构，则采用一体化的思想，对语法结构中的特殊结构进行整合提取，如循环、跳转、逻辑判断等，即忽视其内部过程，只关注该结构的开始与结束，只在开始前与结束后插入探针。为了提高融合的多样性，需要对特殊的语法结构进行特殊的执行探针插入操作，

[0039] (1)若结点结构为循环结构，不插入执行探针；

[0040] (2)若结点结构为条件分支结构，记录逻辑条件，设置执行探针，在每一条分支结构内部的开始位置插入执行探针以获取执行当前分支所需要的触发条件；

[0041] (3)若结点结构为函数调用结构，记录函数调用的相关信息，并进入函数内部，对函数内部语法结构进行分析，将函数作为一个整体，在整体后插入执行探针以获取函数的参数个数、参数类型以及返回值类型；在存在递归调用情况的函数则将其视为循环结构，与代码中存在的其他循环结构的处理方式相同，为避免循环体对融合代码产生不利影响，对所有循环体与递归调用函数均不对其进行执行探针插入；

[0042] (4)若结点结构为普通结构(一般的变量操作)，记录变量信息，直接插入执行探针。

[0043] 算法如下表1所示。

[0044] 表1语法树控制流识别算法

算法 1： 语法树控制流识别算法

输入： 语法树顶层结点 Ast;深度阈值 value
输出： 控制流结点 Node

[0045]

```
Begin
    ID = 0
    procedure AstToFlow(Ast,value)
        if Ast.Chlid.Count <= 0 || value <=0 then do
            Return          //若没有子结点,或超过阈值则停止
        else then do
            foreach ast in Ast.Child
                if type(ast) == 'for' then do
                    Addinfo(ID,'for','')      //记录该点为循环结构
                    ID = ID + 1
                else if type(ast) == 'if' then do
                    Addinfo(ID,'if',ast.element)      //记录该点为分支结构,并记录逻辑条件
                    ID = ID + 1
                else if type(ast) == 'function' then do
                    Addinfo(ID,'function',ast.element)
                    //记录该点为函数调用结构,并记录函数调用的相关信息,并进入函数内部
                    ID = ID + 1
                    AstToFlow(ast,value-1)
                else then do
                    Addinfo(ID,'var',ast.element)      //记录该点为普通结构,并记录变量信息
                end if
            end foreach
        end if
    end procedure
    AstToFlow(Ast,value)
end
```

[0046] 完成代码的遍历之后，需要根据不同的语法结构类型确定简化的控制结构图：对语法结构中的各种操作类型结构进行整合提取，参照C语言代码风格，对do-while，for，while等常规循环类型结构采用一体化表示，不在控制流中体现循环过程，只关注循环的开始与结束，忽视其内部的其他操作，因为在进行融合过程中，循环体内的融合会使融合程序产生预期之外的错误，所以只关注循环开始前与结束后的程序执行流。而针对部分语言中goto类型的跳转指令，则以位置标识符为起始，goto指令为终止为一体结构。对if-else，switch-case等分支条件结构，则采用各分支代码插桩的方式，在每一条条件分支中插入探针语句，获取当前分支下，前方判断条件的元素取值，再通过动态执行的方式，根据探针的执行情况判断哪条分支可用，并进行记录。然而，在具体程序中，大多会存在嵌套的语法结构，针对嵌套的结构，则以外层的语法结构类型为准，最大限度的覆盖每一个分支。

[0047] 由于忽视结点内部的结构，所以可以进行插入执行探针的结点之间不存在嵌套或包含关系，这样不仅可以满足语法树向控制流的转化条件，同时可以有效地避免在进行融合的过程中，因其内部混乱的执行顺序使融合程序产生预期之外的错误。

[0048] 在完成执行探针的插入工作后，通过动态运行母体程序的方式，根据探针的执行情况获取到母体程序源码的执行顺序。在正常情况下，该执行顺序应具有顺序不变性，路径唯一性，结点必经性等特点。

[0049] 确定母体代码的执行流并获取到关键执行结点后，需要将母体代码的功能与目标代码的功能融合在新的程序之中，该程序利用母体程序的复杂性，将目标代码片段隐藏在母体程序的代码之中，并跟随母体程序的执行流进行执行，从静态、动态两个方面对目标程序进行保护。

[0050] 该过程主要思想为：融合需要根据执行流刻画所得到的运行点记录进行结构上的融合。从运行点记录中可以获得每个语法树结点的结构类型，以及所包含固定组成元素变量信息，根据不同的结构类型选择不同的融合方式。

[0051] 为保证切片与融合的正确性，需要在操作过程中满足以下要求：

[0052] (1)目标代码的切分数量不可大于执行点记录中的可用点数量，对目标代码的切片按照语法树进行完整切片，运行点记录中的数量即为可融合的分片数量，对目标代码的程序分片数量不能大于该值，否则目标代码无法完成完整的融合。

[0053] (2)目标代码分片与母体进行融合的顺序需要按照执行点记录中的位置顺序进行；执行点记录描述母体代码的执行顺序，若在融合过程中，目标代码分片在执行过程中，执行顺序没有按照执行点顺序，则会造成目标代码执行失败。

[0054] (3)目标代码分片的融合点优先选择和自身结构类型相似的点进行融合；融合的过程按照“优先选择相同或相似类型的代码结构之间进行融合”的原则，例如母体的分支结构与目标代码分片中的分支结构进行融合，因为相似的代码结构之间更容易满足变量共用，结构共用的融合要求。

[0055] 在满足以上条件的情况下，目标代码切片与母体程序对应部位的融合应根据不同的结构类型选择不同的融合方式，主要有以下三种方式：

[0056] (1)代码直接填充：对于目标代码分片中的普通表达式、循环体等简单语句操作，直接将代码以原形式填充到母体的关键结点位置。

[0057] (2)预混淆后填充：将循环体代码进行控制流扁平化处理，更改原有的代码结构，

但不更改程序语义，为目标代码提供进一步保护。

[0058] (3)相似结构融合：根据目标代码分片中不同的代码结构，使用不同的融合手段在语法结构上进行融合。针对代码中存在的分支结构，例如代码中存在的if-else，switch-case等分支结构，则将母体的分支条件与要保护的代码分支条件取并交集，形成新的分支条件，在分支内容上则重复“代码直接填充”方式中的过程。针对代码中的函数调用、函数定义，通过将函数一对一进行结构融合，将目标代码中的函数定义结构与母体代码中的函数定义结构进行融合，即将目标代码函数定义中的操作转移至母体代码的函数定义中，并添加运行参数，以区分二者功能，保证功能的正确性。

[0059] 进行融合的过程中，为使母体代码与目标代码之间产生逻辑关联，需要首先将母体语法树中的变量引入目标代码中，即构建二者变量间转换关系。根据执行点记录中记录的变量类型与变量值，对目标代码分片中的操作进行等价转换，将目标代码分片构建为不透明谓词，并根据记录将母体中的变量作为不透明谓词的一个元素，构建出与原表达式具有相同语义的融合表达式。以此，目标代码与母体代码之间通过共用变量的方式，在不影响二者语义的情况下，在结构、逻辑上产生关联，增强了二者的紧密程度。

[0060] 具体算法描述见下表2。

[0061] 表2语法树融合算法

[0062]

算法 2： 语法树融合算法

输入： 控制流结点 *Node*，核心脚本分片 Slice
输出： 控制流融合后片段 Ast

```
begin
1   Array<Slice>peice
2   Array<Node>Nodes
```

[0063]

```
3   procedure FuseAst(Node,Slice)
4       if type(Node) == 'if' then do
5           if type(Slice) == 'if' then do
6               Node.condition = Node.condition + Slice.condition      //分支结构中将二者条件取并集
7               Insert(Node,Slice)      //将分片的其他部分插入条件动作中
8               Fuse(Node.element,Slice.element)      //设置变量共用，构建不透明谓词
9           else then do
10              Insert(Node,Slice)
11              Fuse(Node.element,Slice.element)
12          end if
13      else if type(Node) == 'function' and type(Slice) == 'function' then do
14      //若同为函数调用，则将分片函数体与母体函数定义进行融合
15          FuseAst(Node,Slice.Body)
16      else then do
17          Insert(Node,Slice)      //普通融合，并设置变量共用
18          Fuse(Node.element,Slice.element)
19      end if
20  end procedure
21  foreach Node in Nodes
22      foreach Slice in peice
23          FuseAst(Node,Slice)
end
```

[0064] 从而将母体代码的功能与目标代码的功能融合在新程序中，可保证功能完整性的前提下有效隐藏代码自身的逻辑功能。

[0065] 本发明通过使要保护的代码与无害的母体代码进行融合，摒弃传统的混淆方法中对源码形式的过度处理，通过将核心代码隐藏在复杂的母体代码中，分段隐藏具体的代码

行为。在功能完整性上，核心代码通过将自身的控制流结构与母体代码的控制流结构相融合，并选取母体执行流中的关键点或必经点作为融合点，保证了程序的执行正确性与不变性。在静态分析层面，原程序控制流图被破坏，各关键控制流节点加入了新的控制流过程，但主体依然是无害母体的控制流。在动态分析层面，生成的新代码程序在很大程度上依然主要为母体代码的行为，在分析过程中，核心操作分片隐藏在母体代码中，通过变量绑定结构共用的方式融合的代码使母体与核心代码之间存在不易分割的逻辑关系，无法区分母体代码程序与核心代码程序，无害的母体代码为核心代码提供了良好的隐藏效果。能够有效解决传统混淆方法难以对抗自动化分析手段，具有明显混淆特征的问题。并且，该混淆方法适用于大多数编程语言，具有良好的普适性。

[0066] 实施例2：本实施例以C程序控制流图为例，融合前后母体代码与目标代码的控制流图变化如下所示，结果如图2-4所示，其中图2为融合前目标代码控制流图，图3为融合前母体代码控制流图，图4为融合后代码控制流图。

[0067] 目标代码的相关数据在程序初始阶段进行定义，在融合之后，目标代码数据与母体程序数据进行整合，均在程序初始阶段，即控制流图深度为0的位置(最上层基本块以0开始，下文相同)进行初始化，保证程序的数据访问需求。目标代码程序的控制流图的基本块分散在融合之后的程序控制流图中，例如融合后的控制流图中深度为4的其中一个基本块，在语义上包含了目标代码控制流图中深度为3的基本块语义内容，其中不一致的部分属于母体代码的控制流。融合后代码的控制流图与目标代码控制流图在深度和形态上具有较大区别，并且存在相似的部分包含母体代码控制流的部分语句，具有较强的抗同源分析效果。

[0068] 可以明显看出，进行融合之后，母体程序中的关键节点前插入核心代码的分片，在母体代码执行的过程中，核心代码分片分散在母体代码的各个部位，并通过代码变形，改变代码原有的结构，但依照顺序进行执行，隐式的执行核心代码的全部功能。目标代码在与母体融合后，程序的控制流发生了较大变化，在整体的控制流结构上更接近于母体程序的控制流结构，其本身的控制流与母体代码的控制流相结合，并难以区分，增强了程序的抗同源性分析能力，混淆效果更佳。

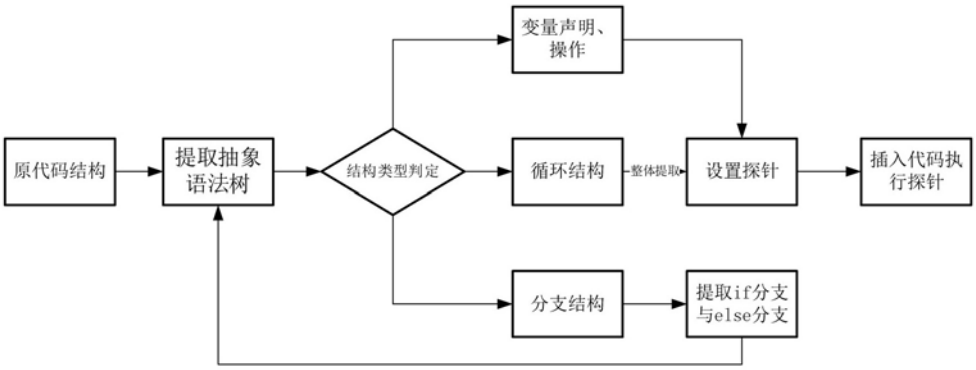

变量声明、操作
原代码结构
提取抽象语法树
结构类型判定
循环结构
整体提取
设置探针
插入代码执行探针
分支结构
提取if分支与else分支

图1

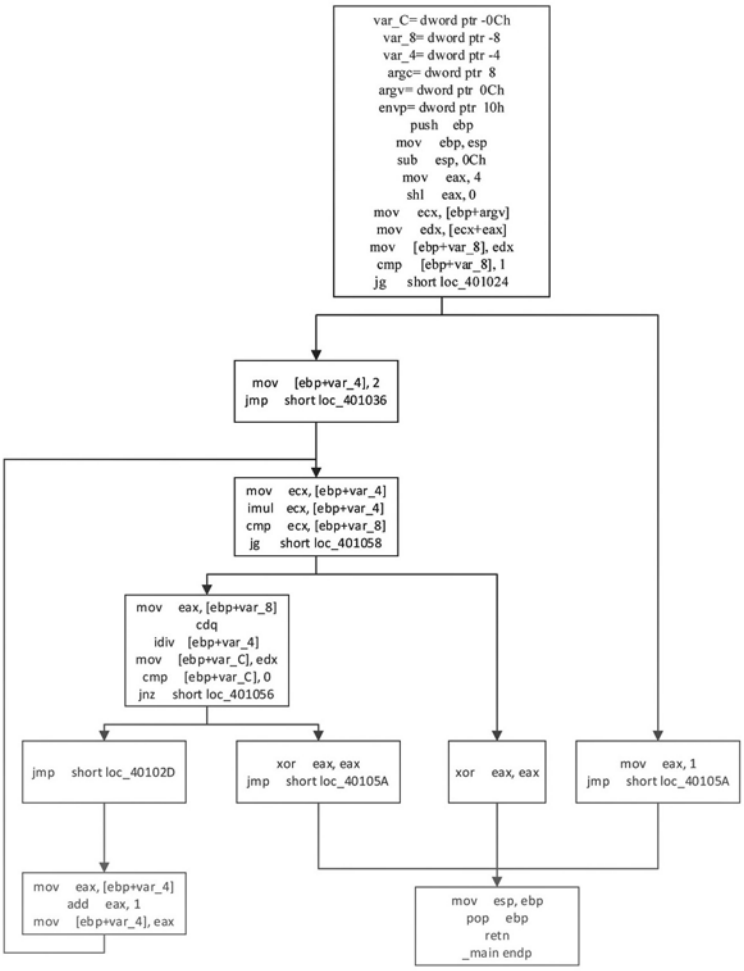

var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
push ebp
mov ebp, esp
sub esp, 0Ch
mov eax, 4
shl eax, 0
mov ecx, [ebp+argv]
mov edx, [ecx+eax]
mov [ebp+var_8], edx
cmp [ebp+var_8], 1
jg short loc_401024
mov [ebp+var_4], 2
jmp short loc_401036
mov ecx, [ebp+var_4]
imul ecx, [ebp+var_4]
cmp ecx, [ebp+var_8]
jg short loc_401058
mov eax, [ebp+var_8]
cdq
idiv [ebp+var_4]
mov [ebp+var_C], edx
cmp [ebp+var_C], 0
jnz short loc_401056
jmp short loc_40102D
xor eax, eax
jmp short loc_40105A
xor eax, eax
mov eax, 1
jmp short loc_40105A
mov eax, [ebp+var_4]
add eax, 1
mov [ebp+var_4], eax
mov esp, ebp
pop ebp
retn
_main endp

图2

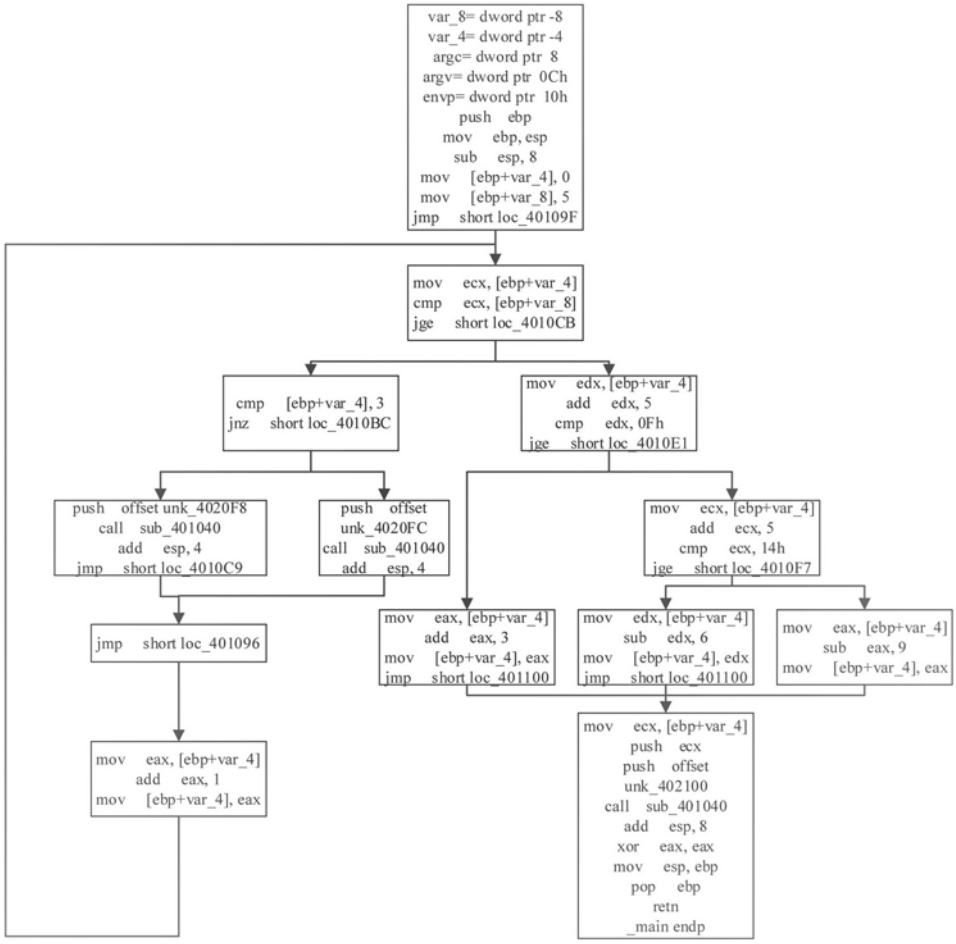

var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
push ebp
mov ebp, esp
sub esp, 8
mov [ebp+var_4], 0
mov [ebp+var_8], 5
jmp short loc_40109F
mov ecx, [ebp+var_4]
cmp ecx, [ebp+var_8]
jge short loc_4010CB
cmp [ebp+var_4], 3
jnz short loc_4010BC
mov edx, [ebp+var_4]
add edx, 5
cmp edx, 0Fh
jge short loc_4010E1
push offset unk_4020F8
call sub_401040
add esp, 4
jmp short loc_4010C9
push offset unk_4020FC
call sub_401040
add esp, 4
mov ecx, [ebp+var_4]
add ecx, 5
cmp ecx, 14h
jge short loc_4010F7
jmp short loc_401096
mov eax, [ebp+var_4]
add eax, 3
mov [ebp+var_4], eax
jmp short loc_401100
mov edx, [ebp+var_4]
sub edx, 6
mov [ebp+var_4], edx
jmp short loc_401100
mov eax, [ebp+var_4]
sub eax, 9
mov [ebp+var_4], eax
mov eax, [ebp+var_4]
add eax, 1
mov [ebp+var_4], eax
mov ecx, [ebp+var_4]
push ecx
push offset unk_402100
call sub_401040
add esp, 8
xor eax, eax
mov esp, ebp
pop ebp
retn
_main endp

图3

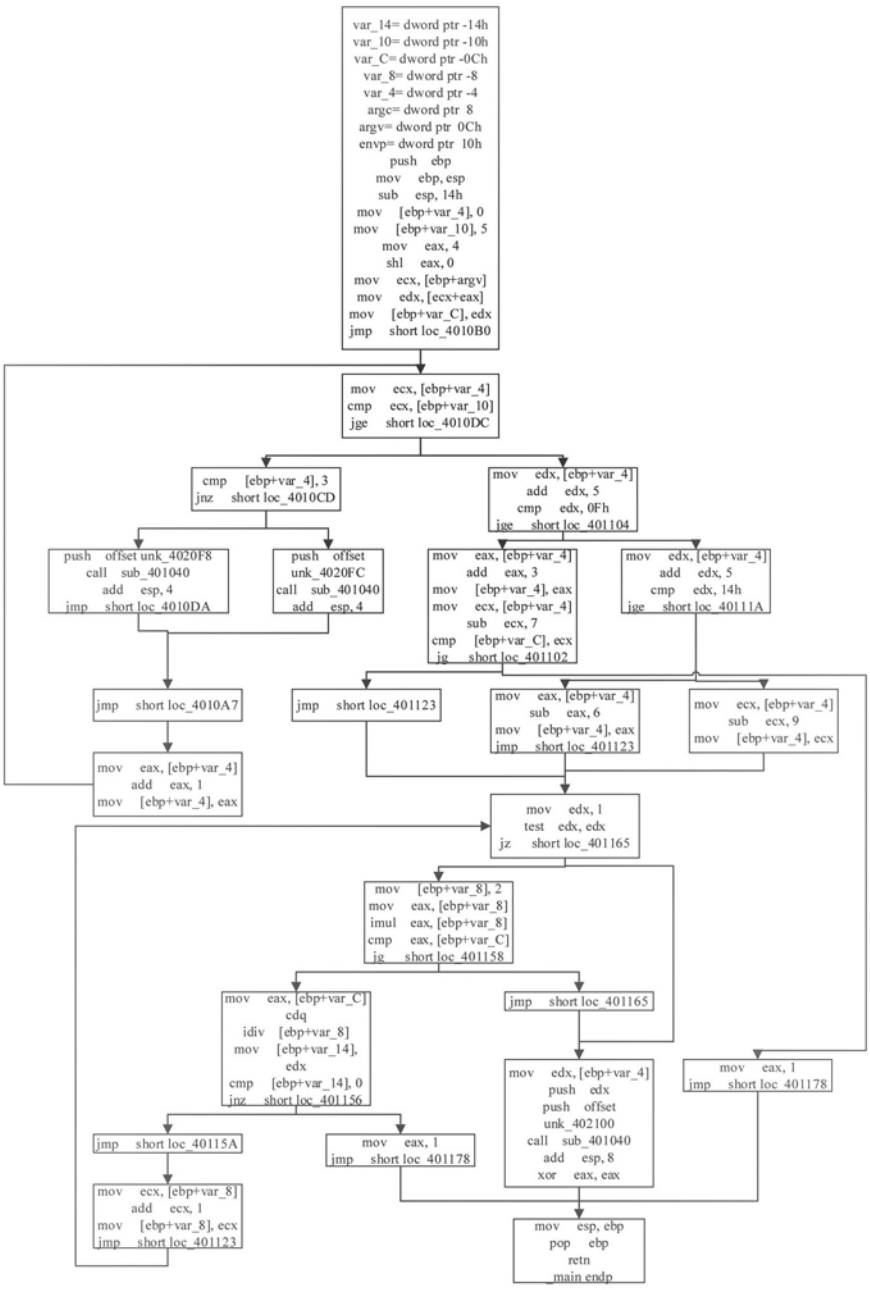

var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
push ebp
mov ebp, esp
sub esp, 14h
mov [ebp+var_4], 0
mov [ebp+var_10], 5
mov eax, 4
shl eax, 0
mov ecx, [ebp+argv]
mov edx, [ecx+eax]
mov [ebp+var_C], edx
jmp short loc_4010B0
mov ecx, [ebp+var_4]
cmp ecx, [ebp+var_10]
jge short loc_4010DC
cmp [ebp+var_4], 3
jnz short loc_4010CD
mov edx, [ebp+var_4]
add edx, 5
cmp edx, 0Fh
jge short loc_401104
push offset unk_4020F8
call sub_401040
add esp, 4
jmp short loc_4010DA
push offset unk_4020FC
call sub_401040
add esp, 4
mov eax, [ebp+var_4]
add eax, 3
mov [ebp+var_4], eax
mov ecx, [ebp+var_4]
sub ecx, 7
cmp [ebp+var_C], ecx
jg short loc_401102
mov edx, [ebp+var_4]
add edx, 5
cmp edx, 14h
jge short loc_40111A
jmp short loc_4010A7
jmp short loc_401123
mov eax, [ebp+var_4]
sub eax, 6
mov [ebp+var_4], eax
jmp short loc_401123
mov ecx, [ebp+var_4]
sub ecx, 9
mov [ebp+var_4], ecx
mov eax, [ebp+var_4]
add eax, 1
mov [ebp+var_4], eax
mov edx, 1
test edx, edx
jz short loc_401165
mov [ebp+var_8], 2
mov eax, [ebp+var_8]
imul eax, [ebp+var_8]
cmp eax, [ebp+var_C]
jg short loc_401158
mov eax, [ebp+var_C]
cdq
idiv [ebp+var_8]
mov [ebp+var_14], edx
cmp [ebp+var_14], 0
jnz short loc_401156
jmp short loc_401165
mov edx, [ebp+var_4]
push edx
push offset unk_402100
call sub_401040
add esp, 8
xor eax, eax
mov eax, 1
jmp short loc_401178
jmp short loc_40115A
mov eax, 1
jmp short loc_401178
mov ecx, [ebp+var_8]
add ecx, 1
mov [ebp+var_8], ecx
jmp short loc_401123
mov esp, ebp
pop ebp
retn
_main endp

图4