浮点数比较分支的混淆方法研究

耿 普* 祝跃飞

(信息工程大学 郑州 450002)

摘 要：针对当前分支混淆方法仅对整数比较分支有效的缺陷，该文分析浮点数二进制表示与大小比较的关系，证明了浮点数二进制区间的前缀集合与浮点数区间内数据之间具有前缀匹配关系。使用哈希函数对前缀集合进行保护，利用哈希函数的单向性实现对抗符号执行，通过哈希值比对替换浮点数比较，提出一种基于前缀哈希值比较的分支条件混淆技术，实现了一种在符号执行对抗和混淆还原对抗上具有较强对抗性的混淆方法。最后，通过实验证和分析，证实了该文提出的混淆方法有消耗小、能够有效对抗符号执行和混淆还原的优点，具备较好的实用性。

关键词：分支混淆；浮点数比较；前缀算法；符号执行

中图分类号：TP311 **文献标识码**：A **文章编号**：1009-5896(2020)12-2857-08

DOI: 10.11999/JEIT190743

An Branch Obfuscation Research on Path Branch Which Formed by Floating-point Comparison

GENG Pu ZHU Yuefei

(*Information Engineering University*, *Zhengzhou* 450002, *China*)

Abstract: For the faultiness that the recent branch obfuscation method is only efficient on branch condition formed by integer comparison. The relations between the binary representation and big or small comparison of floats are analyzed. The idea that the floats in float interval has prefix matching relation with the prefix set which comes from the binary representation interval of the floats is proved. By protecting the prefix set with Hash function, and based on the comparison of prefix-Hash, a new branch obfuscation method which works well on the branch formed by float number comparison is proposed. The new obfuscation method is powerful on symbolic execution combating and obfuscation recovery combating. At last, the obfuscation proposed in this paper is confirmed to be practical, and is useful to be against symbolic execution and obfuscation recovery.

Key words: Branch obfuscation; Float number comparison; Prefix algorithm; Symbolic execution

1 引言

根据商业软件联盟(Business Software Alliance, BSA)的研究报告[1]，软件恶意行为每年在全球造成的经济损失高达6000亿美元,因此与代码恶意行为的斗争的软件恶意行为检测[2]和代码保护是一直是软件安全领域的研究热点。在代码保护的研究中，代码混淆[3]由于具有使用方便、保护效果较好且无需额外硬件辅助等优点，成为了代码保护的重要技术，甚至在特定算法对应的硬件电路混淆保护[4]上也取得了较好的效果。特别是路径分支混淆[5–13]技术，由于能对抗以符号执行[14–17]为基础的软件自动分析技术，大幅提升软件恶意分析的消耗，成为了当前的代码保护方向的研究热点。现有分支混淆的研究成果主要用于对整数比较分支的混淆保护，对浮点数比较分支的混淆方面尚无相关研究，基于此本文通过分析比较浮点数存储方式和大小比较之间的关系，提出了一种针对浮点数比较的分支条件混淆技术，扩大了分支条件混淆的应用范围。本文主要研究内容如下：

(1) 分析了浮点数的大小比较与内存数据之间的关系；并证明了浮点数区间和浮点数对应内存数据的二进制前缀集合之间具有前缀匹配关系。

(2) 提出了一种对浮点数比较分支条件进行混淆的方法。针对浮点数的存储特性与整数具有相似性的特征，把存储浮点数转换为具有相同内存数据的整数进行处理；引入前缀算法压缩分支输入个

收稿日期：2019-09-27；改回日期：2020-05-23；网络出版：2020-07-09
*通信作者： 耿普 23015636@qq.com
基金项目：国家重点研发计划(2016YFB0801601, 2016YFB0801505)
Foundation Items: The National Key R&D Program of China (2016YFB0801601, 2016YFB0801505)

数；使用前缀哈希比较替代浮点数比较，完成分支混淆。

(3) 通过实验证实了对浮点数比较条件的混淆方法的实用性和有效性。

2 基础知识和相关工作介绍

2.1 前缀匹配和前缀算法

前缀匹配的定义：任意两个二进制数$a=a_1a_2\cdots a_m$和$b=b_1b_2\cdots b_n$，如果存在$k(0\le k\le\min(m,n))$使得$a_1a_2\cdots a_k=b_1b_2\cdots b_k$且$a_{k+1}\ne b_{k+1}$，则称$a$和$b$是$k$-bit前缀匹配的。若一个$k$-bit的前缀元素prex与二进制数$a$具有$k$-bit前缀匹配关系，则称$a$与前缀元素prex前缀匹配。

前缀算法是一种从二进制数区间$[a,b]$提取前缀集合S的算法，通过算法提取的前缀集合与区间内数据具有前缀匹配关系。即若$x\in[a,b]$，则在S内必定存在一个元素s_i与x前缀匹配；若y与S内的所有前缀元素均前缀不匹配，则必有$y\notin[a,b]$。

算法1 前缀算法

```
输入：a1a2…an//起始值a的二进制表示
      b1b2…bn//结束值b的二进制表示
输出：PrefixSet//区间的前缀集合
PrefixSet Get_Prefix(a1a2…an, b1b2…bn)
{
  for (int k=1; (k<=n) && (ak==bk); k++)
  {
    if (k==(n+1))
      return { a1a2…an};
  }
  if ((akak+1…an == 00…0) && (bkbk+1…bn == 11…1))
  {
    if (k== 1)
      return {*};
    else
      return {a1a2…ak-1};
  }
  PrefixSet1 = Get_Prefix(ak+1ak+2…an, 11…1);
  PrefixSet2 = Get_Prefix(00…0, bk+1bk+2…bn);
  Return {a1a2…ak-10+PrefixSet1, a1a2…ak-11+PrefixSet2};
}
```

根据前缀算法可知，使用前缀算法对二进制数区间提取的前缀集合，其元素个数上限为$2n$-1个，平均个数为n–2，其中n为输入值的二进制位数。

2.2 分支混淆相关研究介绍

在基于分支控制方式变换的混淆方面，通过隐藏跳转指令对抗约束条件获取来实现抗符号执行。2007年Popov等人[5]提出了一种使用异常处理替代分支控制的代码混淆技术；2011年贾春福等人[6]扩展了该混淆技术，把异常处理从无条件跳转分支混淆扩展到了条件跳转分支。

在基于分支条件计算逻辑变换的混淆方面，主要通过阻碍求解器对约束条件的求解来对抗符号执行。Sharif等人[7]2008年提出了基于哈希保护的相等比较分支的混淆；2011年Wang等人[8]提出了使用未解数学猜想对分支条件计算进行混淆的方法；2014年Zong等人[9]和Ma等人[10]分别提出了使用机器学习把分支条件转换为输入分类器的分支条件混淆方法；2015年王志等人[11]提出了基于保留前缀加密的分支混淆方法，把Sharif等人基于整数等于比较的分支混淆扩展到了大小比较分支；2018年陈喆等人[13]提出了基于随机森林的程序分支混淆技术。

在分支目标地址隐藏上，2015年陈喆等人[12]提出了一种分支判断和程序代码分离的代码混淆方法，隐藏了分支目标地址和输入之间的关系，通过阻碍约束条件获取对抗符号执行。

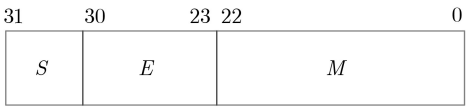
虽然关于分支混淆的研究取得了很多的成果，但是在浮点数比较分支条件的混淆方面尚无相关研究，本文的提出的混淆方法正是对该领域的研究尝试。

3 浮点数比较与分支条件混淆

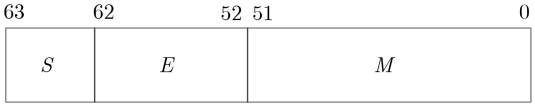
3.1 浮点数的存储和性质

3.1.1 浮点数存储

在大多数计算机内存中，浮点数采用在IEEE 754标准存储。该标准中，浮点数F使用二进制科学计数法表示，即通过符号位S、基数B、指数E和尾数M的组合表示，形如$F=(-1)^S\times M\times B^E$。浮点数的二进制存储如图1，例如单精度浮点数100.0的二进制存储形式为01000010110010000...0，其对应的二进制科学计数法为$(-1)^0\times1.1001\times2^6$，其中1.1001为二进制小数。下面以100.0为例，对浮点数的二进制存储进行介绍。



(a) 单精度浮点数示例



(b) 双精度浮点数示例

图 1 浮点数IEEE 754标准存储示意图

(1) 正数符号位S=0、负数符号位S=1；基数是固定常数2，因此在存储中省略基数。

(2) 尾数M采用规格化表示。对尾数规格化表示，主要是为实现浮点数二进制表示的唯一化。比如100.0，其二进制科学计数法可以表示为1.1001×2^6, 11.001×2^5, 0.11001×2^7, 110.01×2^4等各种形式。在E中存储数据不全为零时，若尾数规格化为1.*的模式，则所有浮点数的二进制科学计数法表示是唯一的。如100.0仅能表示为1.1001×2^6。

(3) 为更好地表示较小数值，E中存储的是浮点数二进制科学计数中指数和127的和。如100.0的二进制科学计数法表示为$(-1)^0\times1.1001\times2^6$，但是$E$中存储的是133=127+6。

(4)指数E在内存中为全0或者全1是一种特殊情况，表示的意义如下：当E中存储的数据为全0，M中存储的数据也为全0时，表示浮点数0.0；当E全0, M非全0时，尾数M不再规格化为1.*的形式，而是0.*的形式；当E全1, M全0时，浮点数值为∞；当E全1, M非全0时，浮点数值为NaN(Not a Number)。

3.1.2 浮点数性质

由于实数元素具有无穷多个，而计算机内浮点数的存储内存使用固定的二进制位数，且还能表示+∞和−∞。即计算机无法表示−∞和+∞之间的所有实数，计算机表示的浮点数具有如下性质：

(1)浮点数是实数的近似表示。计算机中的每个浮点数代表的是一个实数区间，区间内所有实数在计算机中使用该浮点数表示；所有浮点数表示的区间取并集即为(−∞, +∞)。比如[16777219.0, 16777222.0)内所有实数在计算机中的浮点数存储16进制表示均为0x4b800002，即对变量a=16777220.0, b=16777221.0，计算机中认为实数a和b是相等的。

(2)计算机中浮点数表示精度随浮点数绝对值的大小而不同。与零距离越小的浮点数表示精度越高，与零距离越远的浮点数表示精度越低。

3.2 浮点数比较与前缀匹配

由于浮点数是实数的近似表示，而整数是精确表示，因此它们在比较和计算性质上具有差异。但是浮点数的二进制表示形式和大小关系之间的关联与整数是具有相似性的，本节将证明浮点数区间在前缀匹配关系上与整数区间具有相似的性质。

推论1：浮点数的二进制表示中，若符号位为0，则二进制数据的大小关系与浮点数的大小关系一致。

证明：设a和b为任意两个正浮点数，其对应的二进制表示分别为$a_1a_2...a_n$, $b_1b_2...b_n$，且a和b的二进制科学计数法表示$a=M_a\times2^{E_a}$和$b=M_b\times2^{E_b}$。

由二进制科学计数法中的尾数M具有规格化表示可知，若$E_a>E_b$，则M_a必然是具有规格化表示的，而M_b则有可能规格化，也有可能没有规格化。所以有大小关系如下：$a-b>(2\times M_a-M_b)\times2^{E_b}>(2\times1.0-1.111\cdots1)\times2^{E_b}>0$。即$a>b$。

同理若$E_b>E_a$，即$E_a<E_b$,则有$a<b$。

若$E_a==E_b$，则$a-b=2^{E_a}\times(M_a-M_b)$。因此，该情况下若$M_a>M_b$，则$a>b$；若$M_a<M_b$，则$a<b$；若$M_a==M_b$，则$a==b$。

综上，若$a>b$，则必然有$E_a>E_b$或者$M_a>M_b$，在符号位为零时，显然二进制数据$a_1a_2\cdots a_n>b_1b_2\cdots b_n$。若$a_1a_2\cdots a_n>b_1b_2\cdots b_n$，则必有一个最小的$i(i>1$且$i<n)$使得$a_i>b_i$，即必有$E_a>E_b$或者$M_a>M_b$，显然$a>b$成立。同理可知$a_1a_2\cdots a_n==b_1b_2\cdots b_n\Leftrightarrow a==b$; $a_1a_2\cdots a_n<b_1b_2\cdots b_n\Leftrightarrow a<b$。即浮点数大小与二进制存储数据大小具有相同的关系。证毕

推论2：正数浮点数区间[a, b]，通过前缀算法获取[a, b]对应二进制区间的前缀集合S，则浮点数区间[a, b]与S具有前缀匹配关系。

证明：设浮点数a, b的二进制表示为$a_1a_2\cdots a_n$, $b_1b_2\cdots b_n$，二进制数据$a_1a_2\cdots a_n$+00…001表示的浮点数为c。

根据推论1，由$a<b$，有$a_1a_2\cdots a_n<b_1b_2...b_n$；由$a_1a_2\cdots a_n<a_1a_2...a_n$+00…001，有$a<c$。另外，属于二进制区间[$a_1a_2\cdots a_n$,$a_1a_2\cdots a_n$+00…001]且不等于$a_1a_2\cdots a_n$和$a_1a_2\cdots a_n$+00…001的二进制数显然为零个。若存在浮点数$d$使得$d\in(a,c)$，设$d$的二进制表示为$d_1d_2\cdots d_n$，则有$a_1a_2\cdots a_n<d_1d_2\cdots d_n<a_1a_2\cdots a_n$+00…001，这与在[$a_1a_2\cdots a_n$,$a_1a_2\cdots a_n$+00…001]区间内且不等于$a_1a_2...a_n$和$a_1a_2\cdots a_n$+00…001的二进制数个数为0矛盾。因此不存在浮点数d使得$d\in(a,c)$，即[a,c]区间仅包含a,c两个浮点数，即二进制与浮点数按照大小排序后一一对应。因此若浮点数$x\in[a,b]$，即$a_1a_2\cdots a_n<=x_1x_2\cdots x_n<=b_1b_2\cdots b_n$，显然$x_1x_2\cdots x_n$必然与$S$中某个前缀元素前缀匹配。若$x\notin[a,b]$，即$x_1x_2\cdots x_n>b_1b_2\cdots b_n$或者$a_1a_2\cdots a_n>x_1x_2\cdots x_n$，即$S$中不可能有元素与$x_1x_2\cdots x_n$前缀匹配。证毕

若[a, b]区间内的浮点数符号位为1，则对区间内所有浮点数变换符号位为0，负浮点数区间[a, b]变换为正浮点数区间[−b, −a]。对同时包含正数和

负数的浮点数区间，把区间拆分为正数区间和负数区间两个部分，使用两个部分求并集的方式替代原区间。由推论2可知，浮点数的大小比较关系，可以使用前缀匹配关系替代，且当大小比较关系的取值与前缀匹配关系的取值要么同为真，要么同为假，即该替代是保持语程序义下的分支条件变换，这为下一步的分支混淆技术实现打下了坚实基础。

4 分支条件混淆方法

由于浮点数比较存在与整数类似的前缀匹配关系，因此考虑使用基于前缀匹配的分支混淆。2015年王志等人[11]采用IP地址匿名处理[18]和网络功能外包系统[19]等应用中使用的保留前缀加密算法，提出了一种保护大小比较分支的混淆技术，但是保留前缀加密算法在分支混淆中容易遭受还原攻击，因此本文提出了一种基于前缀哈希值比较的分支混淆方法，对浮点数比较的分支进行混淆，提高混淆对抗还原攻击的对抗性。本文提出的混淆方法具体步骤如图2所示，具体步骤如下：

(1) 从分支条件提取浮点数区间，并按照符号位不同，把区间分为正、负两个部分，把负数区间转换为正数区间；然后把两个浮点数区间表示为对应的二进制区间。由于浮点数的二进制存储表示在存在最大值和最小值，且浮点数的二进制表示具有连续性和有序性，因此任意浮点数比较的条件都可以按照符号不同转换为一个或者两个二进制区间。比如对单精度浮点数x, a和b，且浮点数a和b的二进制存储表示分别为$a_1a_2\cdots a_{32}$和$b_1b_2\cdots b_{32}$。如果分支条件为$-b\le x\le a$，则条件对应的浮点数区间为[0, a]和[$-b$, –0], [$-b$, –0]转换为正数区间为[0,b]；二进制区间为[00…0, $a_1a_2\cdots a_{32}$]和[00…0, $b_1b_2\cdots b_{32}$]。

(2) 使用前缀算法获取二进制区间对应的前缀集合S。分别对二进制区间[00…0, $a_1a_2\cdots a_{32}$]和00…0, $b_1b_2\cdots b_{32}$]进行前缀算法，提取前缀集合S_1和S_2。

(3) 对前缀集合进行哈希运算，得到前缀哈希集合。本文采用sha1哈希，分别对前缀集合S_1和S_2中的每个元素求取哈希值，然后在混淆后程序中仅保存前缀集合对应的哈希值集合HS_1和HS_2。

(4)使用前缀哈希相等判断的代码替换原浮点数比较判断的代码。例如对分支条件if(…1.0≤x≤10.0)，使用if(isMatch(x, HS_1, HS_2))替代原分支条件，其中isMatch(x, HS_1, HS_2)是前缀哈希相等判断算法，主要内容如下：若x的二进制表示为$x_1x_2\cdots x_n$，首先记录x_1的值为sign，然后置x_1为0；接着把x二进制表示的前k(k=2, 3, …, n)位作为前缀，对每个前缀求出其哈希值。若sign值为0，使用得到的哈希值分别与HS_1中的值进行比较；若sign值为1，使用得到的哈希值分别与HS_2中的值进行比较；若存在x二进制表示的k位前缀与HS_1或HS_2中的某个哈希值相等，则返回true，表示输入x使得分支条件取值为真；否则返回false，表示x使得分支条件取值为假。

算法2：isMatch(x, HS) //判断输入为x时，分支条件的取值，算法返回值为true或者false

输入：浮点数x，浮点数区间[a, b]对应二进制前缀集合的sha1集合HS1和HS2

输出：x是否属于浮点数区间[a,b]

```
bool isMatch(x,HS1,HS2)
{ char tmp[32] = {'*','*',…,'*'};
    int Ix= *((int *)&x); char sha1out[32][24];
    char sign = (Ix>>(31-i))&1; tmp[0] = sign;
    for(int i=1; i<32; i++){
      tmp[i]=(Ix>>(31-i))&1; sha1out[i]=sha1(tmp,32);
      char sign = tmp[0];
      if(sign == 0)
      {for(int j=0; j<hashNumofHS1;j++)
      { if(sha1out[i]==HS1[j]) return true; }
      }
      else if(sign == 1)
      {for(int j=0; j<hashNumofHS2;j++)
        { if(sha1out[i]==HS2[j]) return true; }
      }
    }
    return false;}
```

本文采用sha1哈希对前缀元素进行保护，若存在两个不同的前缀元素且它们的sha1哈希值相同，则会导致混淆前后程序行为发生变化。但是一方面发现sha1哈希碰撞是一件极其困难的工作；另外在2^{32}个输入集合中出现两个输入使得其sha1哈希相同的概率极低，并且浮点数的二进制前缀中每一位的取值只有0, 1和*共3种选择，极大限制了输入的变化范围，因此sha1哈希在使用32或者64位二进制前缀空间作为输入集合时，可以认为碰撞发生的概率无限趋近于零，即可认为基于前缀哈希匹配的分支混淆不会产生程序语义改变。

5 实验和分析

本文实验基于windows 7系统，CPU是主频为3.20 GHz的Intel I5-3470，内存为6 G。

5.1 混淆消耗分析

5.1.1 空间消耗

如图2所示，在空间消耗方面，从混淆步骤可

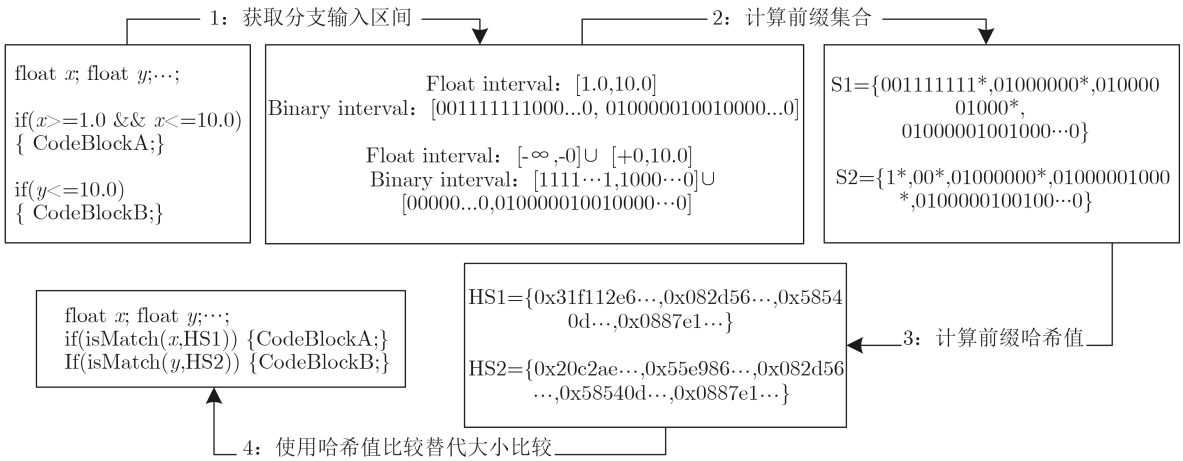


图 2 浮点数比较分支的混淆示意图

知，与混淆前程序相比，混淆后的程序增加了1个哈希算法、1个isMatch算法，以及每个分支混淆增加了1个前缀哈希集合。即哈希算法和isMatch算法为共用空间，所有的被混淆分支均可使用；而哈希前缀集合则每个被混淆分支需要单独占用，根据分支条件对应的浮点数区间不同，其大小具有差异，但是对n位浮点数来说，一个分支最多占用$(2n–1)\times 20$ Byte，平均占用$(n–2)\times 20$ Byte。因此在对m个分支进行混淆时，每个分支平均消耗的空间为$3152/m+20\times(n–2)$。

如表1，对if(1.0≤x≤10.0)进行混淆，由于前缀集合元素个数为4，因此其消耗的空间为3152+4×20=3232 Byte。若对100个单精度浮点数分支进行混淆，则单个分支消耗的空间为3152/100+(32–2)×20=631 Byte，即在被混淆分支数增多时，单个分支混淆消耗的空间比仅对一个分支进行混淆时显著下降。

5.1.2 时间消耗

由于混淆是通过isMatch算法替代原分支中的大小比较判断，而原分支的大小比较判断时间可以忽略不计，因此每个分支条件的混淆消耗时间为isMatch函数消耗的时间。isMatch算法中，消耗时间的代码主要是哈希计算，在n位浮点数比较分支的混淆中，每次分支条件判断需要执行n次哈希计算。通过实验测试，在单精度浮点数比较分支的混淆中，每次isMatch平均消耗时间为0.033 ms，即每次执行混淆后分支需要比混淆前多消耗0.033 ms的时间。

虽然混淆后增加了空间和时间的消耗，但是实现了分支条件混淆，使得逆向攻击者难以通过分支条件获取程序执行逻辑，增强了程序对抗逆向工程的能力。并且分支混淆增加的时间和空间都不是太大，即本文提出的分支混下具有较好的实用性。分支混淆时间和空间的消耗测试数据见表1和表2，其中表2中的数据处理程序是一个浮点数出现在特定区间的频率统计程序。

5.1.3 混淆效率比较

由于当前尚无对浮点数比较分支混淆的研究，因此在执行效率方面，本文方法分别与王志等人[11]、陈喆等人[13]和Ma等人[10]提出的分支混淆方法进行对比。

具体数据见表3，以王志提出的混淆方法在主频为3.20 GHz的Intel I5-3470 CPU和主频为2.66 GHz的Intel Core2 Q9400 CPU的主机上的实验数据作为参照，把本文提出的方法和王志、陈喆、Ma方法在单个大小比较分支上的混淆后执行效率进行比较(在2.66 G的CPU主机上的实验数据中，王志等人方法的数据来源于文献[11]，陈喆等人方法和Ma等人方法的数据来源于文献[13])。由表3数据可知，本文提出的混淆方法与王志等人提出的方法在

表 1　单分支混淆的消耗数据表

分支条件	空间消耗(Byte)			时间消耗(ms)
	解密后前缀数据空间	Sha1算法代码空间	isMatch算法代码空间	
if(1.0≤x≤10.0)混淆后变为：if(isMatch(x, HS1))	4×20=80	2684	468	0.033
if((x≤1.0)||((y>10.0)&&(1.0≤z≤10.0))) 混淆后变为：if(isMatch(x,HS2)||(isMatch (y,HS3) && isMatch(z,HS4)))	(9+8+4)×20=440	2684	468	0.102

注释：(1) 空间消耗中，只有前缀数据占用空间是每个分支混淆需要独占的，其余空间是所有分支混淆共享的空间。(2) HS1, HS2, HS3和HS4表示前缀数据的哈希值集合。

表 2 分支混淆前后程序占用空间和执行时间数据表

	混淆前的数据处理程序	混淆后的数据处理程序
占用空间(Byte)	37376	41472
执行时间(ms)	2	35.6
被混淆分支数(个)	1	
分支执行次数(次)	1000	

执行时间消耗方面大体相当，比陈喆等人方法和Ma等人方法具有更小的执行时间消耗；在混淆空间消耗方面，本文提出的方法与王志等人提出的方法在混淆消耗的空间上大体一致，但是比陈喆等人方法和Ma等人方法具有更小的混淆空间消耗。另外，由于王志等人提出的方法中，保留前缀加密和前缀匹配在同一个程序内实现，因此容易通过保留前缀还原攻击实现混淆还原。因此，综合来看，本文提出的方法相对具有更好的混淆效果。

5.2 对抗性分析

5.2.1 抗混淆还原分析

混淆还原攻击是一种通过混淆后的isMatch函数还原出混淆前分支条件的过程，由于分支混淆过程中采用sha1算法对前缀集合进行了保护，而hash函数具有较好的单向特性。因此通过前缀哈希值恢复前缀，从而实现分支条件恢复的办法显然是行不通的。而通过二分法快速攻击的难度在于找到一个使得分支条件取值为真的初始输入，对于本文的混淆方法，找到一个使得分支条件取值为真的输入，除了通过上下文分析不断压缩分支输入取值区间外，没有比穷尽更好的方法。基于以上两点，本文提出的分支混淆方法在对抗混淆还原上具有较高的强度。

5.2.2 抗符号执行分析

首先，从混淆方法的设计上来看，基于浮点数区间和其二进制表示集合通过前缀算法得到的前缀集合之间具有前缀匹配关系，因此通过Hash函数sha1对前缀集合进行保护，并使用分支输入不同长度前缀的哈希值与前缀集合中每个元素的哈希值进行相等比较，替代浮点数大小比较条件的判断，因此对混淆后程序进行符号执行分析，其难度等同于对一个已知哈希值求取其对应的哈希输入，而哈希函数sha1具有优秀的单向特性，即逆向求取哈希输入是一个困难问题，符号执行技术中对约束条件进行求解的求解器无法通过约束计算出特定哈希值的哈希输入。因此本文提出的方法能够有效对抗符号执行。

从实验数据看，分别使用程序分析工具Angr和KLEE对分支条件if((x≥1.0) && (x≤10.0))混淆后得到的分支if(isMatch(x, HS))进行分析求解。在angr中设置isMatch返回true的分支为found分支，返回false的分支为avoid分支；在klee中则使用klee_make_symbol函数设置x为污染源。结果是Angr返回的found分支数为0，KLEE返回的分支都不是执行到isMatch返回值为true的分支。即Angr和KLEE均没有对isMatch函数攻击成功，无法找到使得isMatch取值为true的分支输入。表4为测试结果。

6 结论

基于浮点数比较分支的分支条件混淆尚无相关研究，本文通过分析研究，证明了浮点数区间内数据与其二进制表示集合得到的前缀集合之间具有前缀匹配关系，从而通过多次前缀哈希值相等比较结果取或计算的方法替代浮点数大小条件的判断，实现了对浮点数比较分支的分支条件混淆。该混淆方法由于使用哈希函数对前缀集合进行保护，从而保证了混淆技术对抗反混淆攻击的强度，也保证了混淆技术对抗符号执行的强度。另外还通过实验和分

表 3 混淆方法执行效率比较

混淆方法	单分支单次执行平均时间消耗(ms)	单分支混淆空间消耗(Byte)	实验主机	分支类型
本文方法	0.033	4×10^3	CPU为Intel I5的主机	浮点数比较
王志方法	0.031	4×10^3	CPU为Intel I5的主机	整数大小比较
王志方法	(11312-1442.7)/(3×10000)=0.329	4×10^3	CPU为Intel Core2 Q9400的主机	整数大小比较
陈喆方法	220	9.8×10^4	CPU为Intel Core2 Q9400的主机	整数大小比较
Ma方法	750	7×10^3	CPU为Intel Core2 Q9400的主机	整数大小比较

表 4 混淆分支的符号执行测试结果

利用符号执行的程序分析工具	执行时间(min)	结果
Angr	80	求解出使得isMatch返回值为真的分支输入值的解个数为0
KLEE	360	共执行593906条指令和 118个分支执行，但求解出使得isMatch返回值为真的分支输入值的解个数为0

析，验证了本文提出的混淆方法能够有效对浮点数比较形成的分支条件进行混淆，在时间和空间消耗较小的前提下隐藏了程序的分支条件信息和执行逻辑，能够有效对抗基于符号执行的自动程序分析和反混淆技术，在一定程度上保证了程序私密算法和信息的安全。

参 考 文 献

[1] Software Management: Security imperative, business opportunity —2018 BSA global software survey. Washington[OL]. https://ww2.bsa.org/-/media/Files/StudiesDownload/2018_BSA_GSS_Report_cn.pdf. 2018.

[2] 梁光辉, 庞建民, 单征. 基于代码进化的恶意代码沙箱规避检测技术研究[J]. 电子与信息学报, 2019, 41(2): 341–347. doi: 10.11999/JEIT180257.
LIANG Guanghu, PANG Jianmin, and SHAN Zheng. Malware sandbox evasion detection based on code evolution[J]. *Journal of Electronics & Information Technology*, 2019, 41(2): 341–347. doi: 10.11999/JEIT180257.

[3] COLLBERG C, THOMBORSON C, and LOW D. A taxonomy of obfuscating transformations[R]. Technical Report 148, 1997.

[4] 张跃军, 潘钊, 汪鹏君, 等. 基于状态映射的AES算法硬件混淆设计[J]. 电子与信息学报, 2018, 40(3): 750–757. doi: 10.11999/JEIT170556.
ZHANG Yuejun, PAN Zhao, WANG Pengjun, *et al*. Design of hardware obfuscation AES based on state deflection strategy[J]. *Journal of Electronics & Information Technology*, 2018, 40(3): 750–757. doi: 10.11999/JEIT170556.

[5] POPOV I V, DEBRAY S K, and ANDREWS G R. Binary obfuscation using signals[C]. The 16th USENIX Security Symposium, Boston, USA, 2007: 275–290.

[6] 贾春福, 王志, 刘昕, 等. 路径模糊: 一种有效抵抗符号执行的二进制混淆技术[J]. 计算机研究与发展, 2011, 48(11): 2111–2119.
JIA Chunfu, WANG Zhi, LIU Xin, *et al*. Branch obfuscation: An efficient binary code obfuscation to impede symbolic execution[J]. *Journal of Computer Research and Development*, 2011, 48(11): 2111–2119.

[7] SHARIF M, LANZI A, GIFFIN J, *et al*. Impeding malware analysis using conditional code obfuscation[C]. Network and Distributed System Security Symposium, San Diego, USA, 2008: 321–333.

[8] WANG Zhi, MING Jiang, JIA Chunfu, *et al*. Linear obfuscation to combat symbolic execution[C]. The 16th European Symposium on Research in Computer Security, Leuven, Belgium, 2011: 210–226. doi: 10.1007/978-3-642-23822-2_12.

[9] ZONG Nan and JIA Chunfu. Branch obfuscation using "Black Boxes"[C]. 2014 Theoretical Aspects of Software Engineering Conference, Changsha, China, 2014: 114–121. doi: 10.1109/TASE.2014.19.

[10] MA Haoyu, MA Xinjie, LIU Weijie, *et al*. Control flow obfuscation using neural network to fight concolic testing[C]. The 10th International Conference on Security and Privacy in Communication Networks, Beijing, China, 2014: 287–304.

[11] 王志, 贾春福, 刘伟杰, 等. 一种抵抗符号执行的路径分支混淆技术[J]. 电子学报, 2015, 43(5): 870–878. doi: 10.3969/j.issn.0372-2112.2015.05.006.
WANG Zhi, JIA Chunfu, LIU Weijie, *et al*. Branch obfuscation to combat symbolic execution[J]. *Acta Electronica Sinica*, 2015, 43(5): 870–878. doi: 10.3969/j.issn.0372-2112.2015.05.006.

[12] 陈喆, 王志, 王晓初, 等. 基于代码移动的二进制程序控制流混淆方法[J]. 计算机研究与发展, 2015, 52(8): 1902–1909. doi: 10.7544/issn1000-1239.2015.20140607.
CHEN Zhe, WANG Zhi, WANG Xiaochu, *et al*. Using code mobility to obfuscate control flow in binary codes[J]. *Journal of Computer Research and Development*, 2015, 52(8): 1902–1909. doi: 10.7544/issn1000-1239.2015.20140607.

[13] 陈喆, 贾春福, 宗楠, 等. 随机森林在程序分支混淆中的应用[J]. 电子学报, 2018, 46(10): 2458–2466. doi: 10.3969/j.issn.0372-2112.2018.10.020.
CHEN Zhe, JIA Chunfu, ZONG Nan, *et al*. Branch obfuscation using random forest[J]. *Acta Electronica Sinica*, 2018, 46(10): 2458–2466. doi: 10.3969/j.issn.0372-2112.2018.10.020.

[14] KING J C. Symbolic execution and program testing[J]. *Communications of the ACM*, 1976, 19(7): 385–394. doi: 10.1145/360248.360252.

[15] 崔宝江, 梁晓兵, 王禹, 等. 基于回溯与引导的关键代码区域覆盖的二进制程序测试技术研究[J]. 电子与信息学报, 2012, 34(1): 108–114. doi: 10.3724/SP.J.1146.2011.00532.
CUI Baojiang, LIANG Xiaobing, WANG Yu, *et al*. The study of binary program test techniques based on backtracking and leading for covering key code area[J]. *Journal of Electronics & Information Technology*, 2012, 34(1): 108–114. doi: 10.3724/SP.J.1146.2011.00532.

[16] BANESCU S, COLLBERG C, GANESH V, *et al*. Code obfuscation against symbolic execution attacks[C]. The 32nd Annual Conference on Computer Security Applications, Los Angeles, USA, 2016: 189–200. doi: 10.1145/2991079.2991114.

[17] BANESCU S, COLLBERG C, and PRETSCHNER A. Predicting the resilience of obfuscated code against symbolic execution attacks via machine learning[C]. The 26th

USENIX Security Symposium, Vancouver, Canada, 2017: 661–678.

[18] FAN Jinliang, XU Jun, AMMAR M H, *et al*. Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme[J]. *Computer Networks*, 2004, 46(2): 253–272. doi: 10.1016/j.comnet.2004.03.033.

[19] 魏凌波, 冯晓兵, 张驰, 等. 基于前缀保持加密的网络功能外包系统[J]. 通信学报, 2018, 39(4): 159–166. doi: 10.11959/j.issn.1000-436x.2018057.

WEI Lingbo, FENG Xiaobing, ZHANG Chi, *et al*. Network function outsourcing system based on prefix-preserving encryption[J]. *Journal on Communications*, 2018, 39(4): 159–166. doi: 10.11959/j.issn.1000-436x.2018057.

耿 普：男，1982年生，博士生，研究方向为信息安全.

祝跃飞：男，1962年生，教授、博士生导师，研究方向为网络空间安全.

责任编辑：马秀强