基于单向保留前缀算法的路径混淆研究

王志，贾春福，刘伟杰，王晓初

（南开大学 信息技术科学学院，天津 300071）

**摘要：**软件在动态执行过程中泄露了大量的路径分支信息，这些路径分支信息是软件内部逻辑关系的二进制表示。在恶意主机环境中，攻击者利用动态监控技术可以详细地记录软件执行过程的路径分支信息，然后利用静态推理技术将路径分支信息用于代码盗用、软件恶意篡改、软件破解、软件盗版等，给软件知识产权保护带来严重威胁。单向函数可以有效地保护基于单一触发条件的路径分支信息，但是单向函数不具有保序性，难以保护基于上下边界判断的路径分支信息。本文将单向函数与保留前缀算法相结合提出了单向保留前缀算法，扩展了单向函数在软件保护中的应用范围，讨论了单向保留前缀算法用于软件路径混淆的可行性和安全性，将攻击者利用软件路径分支信息泄露问题进行攻击的难度等价到逆向计算单向函数的难度。实验结果表明该混淆技术能有效保护各种路径分支信息，额外时间开销和空间开销有限，具有实用性。

**关键字：**软件知识产权保护；路径分支信息泄露；路径混淆；保留前缀加密

中图分类号: **TP393** 文献标志码: **A**

Branch Obfuscation Using One-Way Prefix Preserving Encryption

WANG Zhi, JIA Chun-fu, LIU Wei-jie, WANG Xiao-chu

(College of Information Technical Science, Nankai University, Tianjin 300071, China)

**Abstract：**At run time, a large number of software branching information is leaked. Branching information is the binary representation of the software internal logic. In a malicious host environment, adversaries can dynamically collect the leaked branching information, and use it for code reuse, software malicious tampering, software cracking, software piracy, and so on, which posed a significant threat to software intellectual property protection. One-way hash function can effectively safeguard equal branching conditions, but it is not order-preserving and can’t be used to protect branching information containing unequal trigger conditions, such as greater than or less tan. In this paper, a new branch obfuscation approach using one-way prefix-preserving encryption algorithm is proposed, which extends the protection scope of one-way hash function. . The strength and resilience of the branch obfuscation are discussed. The experimental results show that branch obfuscation could effectively mitigate software’s branching information leaking problem, yet practical in terms of performance.

**Key words:** software intellectual property protection; branching information leaking; branch obfuscation; prefix-preserving encryption

**1. 引言**

据商业软件联盟(BSA)的研究报告显示，代码盗用、软件破解、软件盗版等侵权行为在2010年和2011年给全球软件产业界造成的经济损失高达588亿美元和634亿美元[1][2]。

软件的二进制代码中有大量的逻辑关系需要保护，在软件的执行过程中，这些逻辑关系被二进制代码中的条件跳转指令以路径分支的形式大量泄露，路径信息泄露问题给软件知识产权保护带来了严重的威胁。

逆向工程是软件安全的重要威胁。最近，基于符号执行[3][4]、动态污点分析[5][6]和约束求解[7][8]的软件逆向分析方法受到了研究者的广泛关注，并得到了迅速发展。该方法综合了传统的静态分析和动态分析的优点，利用软件执行过程中的路径信息泄漏问题，实现对软件内部的逻辑关系进行全面而准确地逆向分析，给软件知识产权保护带来了新的威胁。其具体分析过程如图1所示：首先，在虚拟机中动态执行软件，获得软件的二进制执行轨迹；接着，使用符号执行收集并形式化表示执行轨迹中的路径分支条件；然后，利用约束求解技术推理软件的内部逻辑关系。

通过与虚拟机、污点分析、云计算和二进制代码插装等技术的结合，该动静结合的逆向分析技术在分析精度和速度方面都得到了大幅的提升，并已经在软件测试[7-10]、漏洞发掘[11-18]、恶意代码分析[19-23]、协议分析[19][26]和代码复用[27]等安全领域得到广泛的应用。

图1 基于软件路径信息泄露的逆向工程

基于软件路径分支信息泄露问题的逆向工程技术是一柄双刃剑，它既可以用于软件的安全测试，也可以被恶意攻击者用于代码盗用、软件篡改、软件破解以及软件盗版，给软件的知识产权保护带来了严重的威胁。

理想的软件保护模型应该具有单向性，即软件的正向执行过程是准确和高效的，但是逆向工程过程是复杂的，且难以在可接受的时间内完成。

本文将研究具有单向性的软件逻辑关系保护策略，首先，分析了信息安全领域常用的单向函数和保留前缀算法；然后，将单向函数的保护强度和保留前缀算法的转换能力相结合，提出了一种新的路径混淆策略；最后，对基于单向保留前缀加密算法的路径混淆策略进行了安全性分析和性能测试。

# 2. 单向函数和保留前缀算法

## 2.1 单向函数

单向函数被认为是那些正向上容易计算出结果，但是反方向的逆计算却很难的函数。但是，具有完全单向性的函数是否存在，当前还没有被证明，本文所提到的单向函数都是近似的单向函数，其逆向运算的复杂度远远高于其正向计算的复杂度，定义如下：

函数*f*如果能具有以下两个特点，则称*f*为单向函数：

（1）对于属于函数*f*定义域的任一值*x*，都可以轻易地计算出*y* = *f*(*x*)；

（2）对于大部分属于函数*f*值域的任一值 *y*，很难在计算上求出能够满足*f*(*x*)= *y*的值*x*。

在数学中有很多具有单向性的问题，例如著名的背包问题和数论中的因子分解问题等。在信息安全领域中最常用的单向函数有两类:

（1）公开密钥密码体制中所使用的单向陷门函数；

（2）消息摘要中所使用的哈希函数。

哈希函数，通常也被称为单向散列函数或者杂凑函数，可以把任意长度的输入数据打乱并混合，创建一个格式固定的信息摘要，称为哈希值。哈希函数具有很好的单向性，但是不具有保序能力。因此在软件保护中，哈希函数只能用于保护那些等于的逻辑关系，无法对大于或小于的逻辑关系进行保护。

## 2.2 前缀算法

前缀算法能够从一个给定数据空间中找出其前缀特征，即前缀集合。前缀集合具有确定性，能够匹配该前缀集合的数据一定属于该数据空间，否则将一定不属于该数据空间。这样，对一个数据空间的描述既可以利用上下边界来表示，也可以使用该数据空间的前缀特征来表示。

算法1给出了计算给定区间前缀集合算法的伪代码。算法的输入是*n*位二进制数区间的起始值*a*1*a*2*...an*和结束值*b*1*b*2*...bn*，二进制数的左边是高位，右边是低位，输出是该二进制区间的前缀集合。该算法是一个递归算法，首先，找到二进制区间的起始值和结束值的关键位，即从左到右数第一个不相等的比特位，如果没有找到关键位，则说明该二进制区间的上下边界是重合的，它的前缀特征就是该重合的点；然后，如果起始值的关键位*k*右边的各位都是0，而且结束值的关键位*k*右边的各位都是1，则该二进制区间的前缀就是关键位*k*之前的相似部分；接着，如果前两个条件都不满足，则将起始值和结束值的关键位右边的各位分别作为新的边界输入，递归调用前缀算法去分别计算两个子空间的前缀特征；最后，将子空间的前缀特征与关键位*k*左边的相似位进行整合，返回二进制区间的前缀集合。

**算法1.** 求解二进制数区间对应前缀集合的算法

输入：*a*1*a*2*...an* // 一个二进制数区间的起始值

*b*1*b*2*...bn* // 一个二进制数区间的结束值

输出：Set：*Prefix* // 二进制数区间的前缀集合

*Prefix* Search\_Prefix (*a*1*a*2*...an*, *b*1*b*2*...bn*){

// 从第一位开始，找到关键位*k*，*ak < bk*

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 {*a*1*a*2*...ak-*1};

}

*Set\_PrefixA* = Search\_Prefix(*ak+*1*ak+*2*...an*, 11...1 );

*Set\_PrefixB* = Search\_Prefix(00...0, *bk+*1*bk+*2*...bn*)

Return {*a*1*a*2*..ak*-1*0* + *Set\_PrefixA*, *a*1*a*2*...ak-*1*1* + *Set\_PrefixB*};

}

前缀集合的大小对前缀算法的应用有很大的影响，如果前缀集合过大，则前缀匹配过程会大幅增加软件的执行开销。在找到一个数据区间的前缀集合后，前缀的数量会不会和数据区间的大小成正比，即数据区间越大前缀集合是否就越大？事实上，即使对于一个很大的区间，前缀的个数也是有限的，从定理1和定理2可知，无论数据区间的大小如何，前缀个数的最大值只同二进制数的位数*n*相关，前缀数量的最大值为2*n*-2个，平均值约为*n*-2个。

**定理1：**对于任意一个二进制整数区间[*a*1*a*2*...an*, *b*1*b*2*...bn*]（*n*≥2），前缀的个数*p*≤ 2*n*-2。

**定理2：**对于长度为*n*的二进制整数，考虑所有可能的区间[*a*1*a*2*...an*, *b*1*b*2*...bn*]（*n*≥2），前缀平均数为。当*n*很大时，前缀的平均个数趋近于*n*-2。

可见，对于软件二进制代码中常用的32位整数，无论整数区间的大小如何，前缀的个数最多为62个，平均约为30个。

**2.3 保留前缀的加密算法**

保留前缀加密最早应用于IP地址匿名化[28]：在分析互联网信息流动轨迹（internet traffic trace）时，将待分析网络包中明文IP地址以保留前缀的方式映射为32位密文IP地址，使得相同前缀下的明文IP与其对应的密文IP具有相同前缀，防止发送者和接收者地址信息的泄漏，既保护商业和个人信息隐私，又保留了网络结构的节点分布，不会降低分析的精确度。

Li等[29]将保留前缀加密应用于外包数据库中密文数据的直接范围查询：首先，将待查询区间映射为某二进制前缀集合；然后，对集合中所有二进制串进行保留前缀加密，以隐藏查询范围；最后，在密文数据库中检索到与密文前缀匹配的数据作为该次检索结果。该方法转化密文区间查询为前缀匹配判定，在一定安全性前提下能够实现对密文数据的高效范围查询。

保留前缀加密算法的定义：假设两个*n*位的二进制整数*a*和*b*（*a=a*1*a*2*...an*，*b=b*1*b*2*...bn*）的前*k*位都相同，而*k*+1位不同。如果一个加密函数*Ep*，能够保证加密后的*Ep*(*a*)和*Ep*(*b*)的前*k*位仍然相同而第*k*+1位仍然不同，则称加密函数*Ep*是保留前缀的。

保留前缀加密的一般形式定义如下：**给定一个明文*a*=*a*1*a*2*...an*和函数*f*，*f*是任意的可以接收输入*a*的函数，密文*a*´=*a*1´*a*2´*...an*´通过如下的方式计算得到：**

**（1）给定*a*1´为一个常数（0或1）**

**（2）*ai*´*= ai* ⊕ *f*(*a*1*a*2*...ai-*1), *i* = 1, 2, ..., *n***

**假设两个*n*位二进制整数***a*和*b*（*a=a*1*a*2*...an*，*b=b*1*b*2*...bn*）的前*k*（*k* < *n*）位相同，第*k*+1位不同。

当*i*≤*k*时，***a*和*b*的前*k*位相同，所以*a*1*a*2*...ai-*1和*b*1***b***2*...bi-*1**相同。因为函数的输入相同，***f*(*a*1*a*2*...ai-*1)和 *f*(*b*1***b***2*...bi*-1)的输出值也相等，而且，*ai*与*bi*也是相等的，所以， *ai*⊕*f*(*a*1*a*2*...ai-*1)与*bi*⊕*f*(***b*1*b*2*...b****i-*1)的计算结果相同，即*ai*´等于*bi*´；**

**当*i* = *k* + 1时，因为*a*1*a*2*...ai-*1和*b*1***b***2*...bi-*1是**相等的，所以***f*(*a*1*a*2*...ai-*1)和 *f*(*b*1***b***2*...bi*-1)的值也是相等的。但是，*ai*与*bi*不相等，因此，*ai*⊕*f*(*a*1*a*2*...ai-*1)与*bi*⊕*f*(***b*1*b*2*...b****i-*1)的计算结果肯定是不相等的，即*ai*´不等于*bi*´。**

**由上面的证明可知，该方案符合保留前缀加密算法的定义，而且，函数*f*是可以变换的，因此，通过函数*f*可以向保留前缀加密算法中引入新的特性，例如单向性。**

# 3. 基于单向保留前缀加密的路径混淆

由于哈希函数不具有保序性，因此，只能用来保护等于关系的路径分支条件，不能用于保护不等关系的路径分支条件。当路径分支条件是判断某个变量值是否属于一个区间时，Sharif等人[30]提出可以将这种不等关系转换成一组相等关系的集合，将该变量与区间内所有值一一进行比较，然后再使用哈希函数对每个相等关系进行保护。当区间较大时，该方法的时间开销和空间开销将非常大，例如，在二进制代码中一个整数的取值范围是[0～232-1]，那么一个简单的路径分支条件32≤*x*≤111将会被替换成79个相等关系的路径分支条件。每一个相等关系对应一个MD5值，一个MD5值占16个字节。该路径分支条件最少要占用79×16=1264个字节，输入*x*的MD5值要跟集合中的MD5值一一比较，增加了软件运行的空间和时间开销。

保留前缀算法能够将一个区间转换为一个前缀集合，进而将不等关系的判断转换为判断输入值是否匹配不等关系所对应的前缀集合，既增加了攻击者对路径分支条件逆向分析的复杂度，又有效地减少了软件的空间开销和时间开销。

从上一部分知道，判定一个数是否在一个区间内可以转化为判定这个数的前缀是否匹配这个区间对应的前缀集合。但是这个过程是可逆的，如果攻击者获得了前缀集合，就能推理出前缀集合对应数据区间的上下界。因此，需要对前缀集合进行保护，即对前缀进行加密。

软件知识产权保护需要的是一种高效的单向加密方法，哈希函数正是具有这种良好的单向性。单向保留前缀算法的基本思想是把某种经典的哈希函数嵌入到保留前缀加密算法中，使保留前缀加密算法具有单向性，如图2所示。保留前缀加密中的函数*f* 与单向哈希函数相结合，*f*(*a*1*a*2...*ai*)变成了*Γ*(*Hash*(*a*1*a*2...*ai*))，其中*Γ*表示取最低有效位。上述加密方法综合了哈希函数的单向性和保留前缀加密的模式匹配，实现了对软件路径信息的有效保护。



**图2 单向保留前缀加密算法**

表1是对哈希函数、保留前缀加密算法和单向保留前缀加密算法的对比。第一行和第二行是比较次数的最高值和平均值，第三行是算法的单向性。

表1 三种算法的属性对比（*n*为二进制数的位数）

|  |  |  |  |
| --- | --- | --- | --- |
|  | **哈希函数** | **保留前缀加密** | **单向保留前缀加密** |
| 最高值 | 2*n*-1 | 2(*n*-1) | 2(*n*-1) |
| 平均值 | 2*n* | *n*-2 | *n*-2 |
| 单向性 | √ | × | √ |

对于输入为*n*位的二进制整数区间，单向哈希函数的比较次数与区间的大小成正比。保留前缀加密的比较次数与输入的二进制数的位数成正比，因此，其时间开销和空间开销远远小于单向哈希函数，但是，保留前缀加密不具有单向性。单向保留前缀加密算法结合前两种算法的单向性和低开销的优点，具有良好的应用性。

基于单向保留前缀算法的路径混淆过程分为4个步骤，如图3所示。首先，从软件代码中提取大于和小于的路径分支条件，这些分支条件是单向哈希函数不能直接进行保护的；接着，计算能够满足这些不等关系的数值区间，其中MAX和MIN分别表示*x*取值范围中的最大值和最小值；然后，利用单向保留前缀加密算法找到数值区间的前缀集合；最后，将路径分支条件替换成前缀匹配操作。



图3 基于单向保留前缀加密算法的路径混淆过程

# 4. 安全性分析和性能测试

## 4.1 安全性分析

软件路径分支条件的保护要求攻击者即使能够进行多次用例测试，也无法根据运行结果准确推断分支条件。本文通过将单向保留前缀加密算法作用于分支条件对应的数值区间，形成一个或多个前缀集合，从而将输入数值与分支条件的大小关系比较转化为对其前缀的判定，以达到保护路径分支条件的目的。Amanatidis等人[31]首先提出了保留前缀的加密方案。他们通过一个真实的保留前缀方案（real PPE scheme）和一个理想的保留前缀对象（ideal PPE object）定义了安全目标，这里，理想保留前缀对象是一个随机取自于所有保留前缀函数集合中的加密函数。如果，真实的保留前缀方案是“安全的”，那么其与一个理想的随机保留前缀函数应是计算不可区分的。Xiao等人[32]已经证明本文使用的保留前缀加密方案是“安全的”，即与一个理想的随机保留前缀函数是计算不可区分的。理想保留前缀函数在保留前缀的基础上，密文由于保留前缀函数的随机选取而也具有随机性。单向保留前缀方案与理想保留前缀函数的最大相似性，使其在满足不改变程序执行路径的前提下，能够最大限度地保护分支条件信息。

## 4.2 抗逆向分析效果

本部分将使用最先进的逆向分析工具BitBlaze来测试路径混淆策略的抗逆向分析效果。实验中我们选择了一个简单的端口检测程序，如图4所示，该程序只有一个判断条件，用于判断端口号是否在[32,111]这个数值区间。BitBlaze可以对图4中的简单样本进行快速地逆向分析，并精确地计算出合法端口号的范围为[32,111]。



图4 简单的端口检测程序

实验中路径混淆使用MD5算法作为单向函数。表2列出了程序混淆前后BitBlaze对执行轨迹的分析结果，其中包括轨迹中的指令数、跳转指令数、条件跳转指令数、STP文件大小、约束关系数量和约束方程节点数量。

我们可以发现混淆后的执行轨迹明显变长，指令数量增加了200多万条，该路径混淆策略的执行开销还是比较大的，具体的执行开销将在4.3节中详细分析。轨迹中的跳转指令和条件跳转指令的数量反映了程序路径信息的复杂度，从表2中可知，路径混淆后的程序控制流明显复杂了，路径分支数比混淆前增加了大约20万个，即在程序的控制流图中新增了20万个节点。通过对程序执行轨迹的分析，BitBlaze会得到程序的路径约束关系，并用约束求解工具将STP的语法格式记录在文件中。表2的数据显示，混淆后STP文件的大小增加了3倍左右，其中的约束关系数量也增加了1倍左右，因此，路径混淆后的约束关系的复杂度明显增加。表2的最后一列是STP求解过程中所使用的节点数量，节点数直接反应了约束方程的复杂度，节点数越多约束方程越复杂，也就需要使用更多的内存空间和计算时间，数据显示路径混淆后的约束方程增加了7000多个节点，并且STP无法对约束关系进行有效的求解。

实验表明，基于单向保留前缀加密算法的路径混淆策略有效地阻止了逆向工程工具的逆向推理，保护了软件的路径信息，但是，该策略也增加了软件的执行开销。

## 4.3 软件开销测试

基于单向保留前缀加密算法的路径混淆策略会增加软件的体积和执行开销，本节在一个简单的程序样本上对路径混淆的开销进行了测试。测试样本如图5所示，输入为月份和日期两个变量，代码中有两个路径分支条件，共3条不同的执行路径，分别触发行为behavior\_a()、behavior\_b()和behavior\_c()。

实验过程中，程序会随机地选择10000组月份和日期数据输入给测试样本，然后，利用操作系统提供的GetTickCount()函数，计算程序执行过程所使用的时间，单位是毫秒。路径混淆过程分别选择了5种单向哈希函数：CRC32、MD5、SHA1、SHA256和SHA512。为了避免不必要的干扰，实验将重复3次，并计算时间开销的平均值，实验结果如表3所示。



图5 基于日期的路径分支条件

表3的第一行数据是路径混淆前样本的执行时间和文件大小，从表中数据可见，它的执行时间是最少的。第二行数据是采用了CRC32单向哈希函数混淆后样本的执行时间和文件大小，经过3次测试，平均执行时间增加了约8.4秒，是混淆前的6.8倍。因为二进制条件跳转指令的执行时间非常短，可以忽略不计，所以增加的8.4秒是大约2万次输入与前缀集合进行匹配的时间开销，平均每次前缀匹配的时间不到0.5毫秒。随着单向哈希函数复杂度的增加，程序的执行时间也在不断的增加，其中基于最复杂的SHA512单向哈希函数的路径混淆大约增加了15.7倍的程序执行时间，平均每次前缀匹配要花费1毫秒左右的时间。

如表3所示，路径混淆后程序体积的增加并没有像执行时间那样增加了10倍左右，体积仅仅增加了30%到50%，而且体积并不会随着被混淆的路径分支数的增长而快速增长。因为实现单向保留前缀加密的函数可以被重复调用，而且，定理1和2也证明了前缀的数量是有限的，所以，体积的增长也是有限的，不会像Sharif等人[28]提出的混淆方案那样无限地增长。去除引入单向函数所增加的程序体积，每混淆一个路径分支条件大约要增加1000个字节。

基于单向保留前缀算法的路径混淆策略的混淆强度很高，从实验数据上看，其引入的时间和空间开销是有限的。

表2 路径混淆前后BitBlaze分析结果对比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **测试程序** | **指令数** | **跳转指令** | **条件跳转指令** | **STP文件** | **约束关系** | **约束方程节点数** |
| port\_check | 20338 | 3687 | 2672 | 506KB | 49 | 2957 |
| obf\_port\_check | 2361212 | 286629 | 200319 | 1956KB | 79 | 10595 |

表3 基于单向保留前缀加密算法的路径混淆的开销

（时间单位：毫秒；长度单位：字节）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **单向函数** | **时间1** | **时间2** | **时间3** | **平均时间** | **文件长度** |
| Original | 2235 | 1078 | 1015 | 1442.7 | 180272 |
| CRC32 | 10171 | 9797 | 9484 | 9817.3 | 237568 |
| MD5 | 11328 | 10094 | 11250 | 10890.7 | 249908 |
| SHA1 | 11562 | 10656 | 11718 | 11312 | 258103 |
| SHA256 | 12234 | 10984 | 12687 | 11968.3 | 258102 |
| SHA512 | 21907 | 24250 | 21875 | 22677.3 | 282678 |

# 5. 相关工作

软件路径混淆策略的思路有两种：一是增加攻击者收集路径信息的难度；二是增加攻击者分析路径信息的难度。

增加攻击者收集路径信息的难度可以通过将软件的路径分支条件隐藏在非跳转指令的副作用中，例如，贾春福等人[33]通过隐式的CPU指令副作用替换了显式的二进制条件跳转指令，由于能够产生异常的CPU指令和执行环境非常多，攻击者要是穷举软件二进制代码所有可能的异常情况是不现实的，所以极大地增加了攻击者收集软件路径信息的难度。

Falcarin等人[34]和Ceccato等人[35]提出了基于代码移动性的二进制代码混淆策略。在不可信主机环境中的代码是不完整的，在软件执行过程中需要与远程的可信实体进行频繁的代码交互。可信实体是不受攻击者控制的，这样有效限制了攻击者对代码的可见度，增加了攻击者收集路径信息的难度。但是，该方法需要大量的网络开销，二进制代码的执行效率也受到了影响。

增加攻击者分析路径信息的难度是指即使攻击者可以轻易地收集到软件的路径信息，但是要想通过这些泄露的路径信息还原软件逻辑关系是难以完成的。Sharif等人[30]利用单向哈希函数对路径分支条件进行加密，但是哈希函数不具有保序性，只能用于保护软件中的等于关系，而无法保护大于和小于关系。王志等人[36]将未解数学难题引入到软件的路径信息中，使攻击者逆向求解路径分支信息的难度等价到求解未解数学猜想的难度。

# 6. 结论

单向哈希函数具有很好的单向性，但是不具有保序性，只能用于保护等于的路径分支条件。保留前缀算法能够将一个数据区间转换成该区间的前缀集合，将区间的范围查询转换成前缀匹配操作。本文将保留前缀算法与单向哈希函数相结合，提出了基于单向保留前缀算法的路径混淆策略，将路径分支条件中的大于或者小于关系的判断转换为前缀匹配操作，扩展了单向哈希函数在代码混淆中的应用范围。路径混淆后，攻击者逆向分析软件路径分支条件的难度等同于根据输出结果逆向分析哈希函数的输入值的难度，有效缓解了软件路径信息泄露问题所造成的安全威胁。实验结果表明该混淆技术能有效保护软件的路径信息，增加的时间开销和空间开销有限，具有实用性。

# 参考文献

[Eighth Annual BSA and IDC Global Software Piracy Study](http://portal.bsa.org/globalpiracy2010/index.html" \t "_blank), 2011. http://portal.bsa.org/globalpiracy2010/downloads/study\_pdf/2010\_BSA\_Piracy\_Study-Standard.pdf

1. [Ninth Annual BSA Global Software Piracy Study](http://portal.bsa.org/globalpiracy2010/index.html" \t "_blank), 2012. http://portal.bsa.org/globalpiracy2011/downloads/study\_pdf/2011\_BSA\_Piracy\_Study-Standard.pdf

KING J. A new approach to program testing[C]. Proceedings of the international conference on Reliable software. New York, NY: ACM, 1975. 228-233

KING J. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7):385~394

NEWSOME J, SONG D. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C]. Proceedings of the 12th Network and Distributed Systems Security Symposium(NDSS2005). Rosten,VA: Internet Society, 2005.

KANG M, MCCAMANT S, POOSANKAM P, et al. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation[C]. Proceedings of the 18th Network and Distributed System Security Symposium(NDSS2011). Rosten,VA: Internet Society, 2011.

DUTERTRE B, MOURA L. A Fast Linear-Arithmetic Solver for DPLL(T)[C]. Proceedings of the 18th international conference on Computer Aided Verification. Berlin: Springer, 2006. 81-94

GANESH V, DILL D. A Decision Procedure for Bit-Vectors and Arrays[C]. Proceedings of the 19th international conference on Computer Aided Verification. Berlin: Springer, 2007. 519-531

GODEFROID P, LEVIN M Y, MOLNAR D. Automated whitebox fuzz testing[C]. Proceedings of the Network and Distributed System Security Symposium. Rosten, VA: Internet Society, 2008. 1-11

CADAR C, ENGLER D. Execution generated test cases: How to make systems code crash itself[C]. Proceedings of Int SPIN Workshop. Berlin: Springer, 2005. 2-23

CADAR C, ENGLER D, ENGLER D. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs[C]. Proceedings of USENIX OSDI’08. Berkeley, CA: USENIX, 2008.

LEE G, MORRIS J, PARKER K, et al. Using symbolic execution to guide test generation[J]. Software Testing, Verification & Reliability, 2005, 15(1):41-61

BRUMLEY D, POOSANKAM P, SONG D, et al. Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications[C]. Proceedings of IEEE Symposium on Security and Privacy. New York, NY: IEEE, 2008. 143-157

MOLNAR D, LI X C, WAGNER D A. Dynamic test generation to find integer bugs in x86 binary linux programs[C]. Proceedings of USENIX Security Symposium. Berkeley, CA: USENIX, 2009. 67-82

FELMETSGER V, CAVEDON L, KRUEGEL C, et al. Toward Automated Detection of Logic Vulnerabilities in Web Applications[C]. Proceedings of the 19th USENIX Security Symposium. Berkeley, CA: USENIX, 2010. 143-160

WANG T, WEI T, LIN Z, et al. IntScope: Automatically Detecting Integer Overflow Vulnerability in X86 Binary Using Symbolic Execution[C]. Proceedings of the 16th Annual Network and Distributed System Security Symposium. Rosten VA: Internet Society, 2009.

BRUMLEY D, NEWSOME J, SONG D, et al. Towards automatic generation of vulnerability-based signatures[C]. Proceedings of 2006 IEEE Symposium Security and Privacy. Piscataway, NJ: IEEE, 2006. 2-16

BRUMLEY D, WANG H, JHA S, et al. Creating Vulnerability Signatures Using Weakest Preconditions[C]. Proceedings of the 20th IEEE Computer Security Foundations Symposium. Piscataway, NJ: IEEE, 2007. 311-325

CABALLERO J, LIANG Z, POOSANKAM P, et al. Towards Generating High Coverage Vulnerability-Based Signatures with Protocol-Level Constraint-Guided Exploration[C]. Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. Berlin: Springer, 2009. 161-181

CHO C, BABIĆ D, POOSANKAM P, et al. MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery[C]. Proceedings of the 20th USENIX Security Symposium. Berkeley, CA: USENIX, 2011.

BRUMLEY D, HARTWIG C, KANG M, et al. BitScope: Automatically Dissecting Malicious Binaries[R]. School of Computer Science, Carnegie Mellon University, Technical Report CS-07-133, 2007

BRUMLEY D, HARTWIG C, LIANG Z, et al. Automatically Identifying Trigger-based Behavior in Malware[M]. In: Lee W, Wang C, Dagon D, eds. Botnet Detection. Berlin: Springer, 2008. 65-88

COMPARETTI P M, SALVANESCHI G, KIRDA E, et al. Identifying Dormant Functionality in Malware Programs[C]. Proceedings of the 2010 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2010. 61-76

MOSER A, KRUEGEL C, KIRDA E. Exploring Multiple Execution Paths for Malware Analysis[C]. Proceedings of IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2007 231-245

YIN H, SONG D, EGELE M, et al. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis[C]. Proceedings of ACM Conference on Computer and Communication Security. New York, NY: ACM, 2007. 116-127

NEWSOME J, BRUMLEY D, FRANKLIN J, et al. Replayer: Automatic protocol replay by binary analysis[C]. Proceedings of the 13th ACM CCS. New York, NY: ACM, 2006. 311-321

CABALLERO J, JOHNSON N, MCCAMANT S, et al. Binary Code Extraction and Interface Identification for Security Applications[C]. Proceedings of the 17th Annual Network and Distributed System Security Symposium. Rosten VA: Internet Society, 2010.

XU J, FAN J, AMMAR M, et al. Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme[C]. Proceedings of the 10th IEEE International Conference on Network Protocols. Piscataway, NJ: IEEE, 2002. 280-289

LI J, OMIECINSKI E. Efficiency and security trade-off in supporting range queries on encrypted databases[C]. Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security. Berlin: Springer, 2005. 69-83

SHARIF M, LANZI A, GIFFIN J, et al. Impeding malware analysis using conditional code obfuscation[C]. Proceedings of the Network and Distributed System Security Symposium. Rosten,VA: Internet Society, 2008. 321-333

AMANATIDIS G, BOLDYREVA A, O'Neill A. Provably-secure schemes for basic query support in outsourced databases[C]. Proceedings of the 21st annual IFIP WG 11.3 working conference on Data and applications security. Berlin: Springer-Verlag, 2007. 14-30

XIAO L, YEN, I. Security Analysis and Enhancement for Prefix-Preserving Encryption Schemes [R]. Department of Computer Science, University of Texas at Dallas, Technical Report UTDCS-04-12, 2012

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．计算机研究与发展，2011，Vol.48(11)：2111-2119

JIA C, WANG Z, LIU X, et al. Branch Obfuscation: An Efficient Binary Code Obfuscation to Impede Symbolic Execution. Journal of Computer Research and Development, 2011, Vol.48(11):2111-2119

FALCARIN P, CARLO S, CABUTTO A, et al. Exploiting Code Mobility for Dynamic Binary Obfuscation[C]. Proceedings of the 2011 World Congress on Internet Security, Piscataway, NJ: IEEE, 2011. 114-120

CECCATO M, TONELLA P. CodeBender: Remote Software Protection Using Orthogonal Replacement[J]. IEEE Software, 2011, 28(2): 28-34

WANG Z, MING J, JIA C, et al. Linear Obfuscation to Combat Symbolic Execution[C]. Proceedings of the 16th European Symposium on Research in Computer Security (ESORICS 2011), LNCS 6879, Leuven, Belgium, September 2011. 210-226