广西科学院学报
Journal of Guangxi Academy of Sciences

2010,26(4):420~422
Vol. 26, No. 4 November 2010

基于 Java 多线程隐藏数组下标变换表达式的代码迷惑算法

Based on Java Multi - Thread Code Obfuscation via Hiding the Transformation of Subscripts of the Array

刘　九,林孔升,尚汪洋,蔡德霞
LIU Jiu, LIN Kong-sheng, SHANG Wang-yang, CAI De-xia

(广西大学计算机与电子信息学院,广西南宁　530004)
(School of Computer, Electronics and Information, Guangxi University, Nanning, Guangxi, 530004, China)

摘要:对数组下标变换表达式进行预处理,使得表达式能够并行处理后,通过对线程控制,并行求解变换表达式的值,实现一种隐藏数组下标变换过程的代码迷惑算法。该算法所处理的迷惑代码能够很好地抗击源代码静态分析和基于源代码植入反迷惑攻击。

关键词:代码迷惑　多线程　下标　表达式
中图法分类号:TP309.7, TP338.6　　**文献标识码**:A　　**文章编号**:1002-7378(2010)04-0420-03

Abstract: The transformation of array subscripts is pretreated so the subscripts can be computed parallelly. Then the value of the transformed subscripts array can be obtained by controlling the execution of the program. The obfuscation algorithm can prevent the deobfuscation technique based on the static analysis of programs and by using program instrumentation.

Key words: code obfuscation, multi-thread, subscript, expression

代码迷惑技术的研究开始于 20 世纪 90 年代末,目前尚处于有待于成熟阶段[1]。但是随着人们对软件安全问题重视程度和产权保护意识的增强,以及代码迷惑技术无需硬件支持的优点[2],代码迷惑技术日益受到人们的青睐,形成了研究代码迷惑的热潮。Barak[3]认为世界上并不存在完美的迷惑器,但是在后续的研究中他承认代码迷惑有它自身价值。Collerg 等人定义代码迷惑的概念为:通过对程序代码进行迷惑变换,这种变换是在不影响程序执行结果的情况下,使得程序的复杂度和理解难度大大增加,从而使得攻击者不能正确理解程序和还原出迷惑前的源代码,以防止恶意主机问题[1]。代码迷惑技术归纳为 4 大类:版面布局迷惑、数据迷惑、控制流迷惑与预防迷惑[1]。这些迷惑技术都可以设计数组下标的变换,尤其在控制流迷惑技术中,可以通过对下标的复杂变换来增大程序的复杂度和理解难度[4,5]。通过对数组下标变换来隐藏程序的实际控制流的技术,在代码迷惑领域应用广泛,因此研究隐藏数组下标的变换过程显得尤为重要。本文提出一种基于 Java 多线程的隐藏数组下标变换过程的代码迷惑算法,并通过实例分析说明该算法实现的过程。

收稿日期:2010-09-16
修回日期:2010-10-15
作者简介:刘　九(1985-),男,硕士研究生,主要从事信息安全研究。

1　基于 Java 多线程代码迷惑算法

在数组下标的传统变换过程中,即使非常复杂的变换表达式,攻击者也轻而易举的通过静态分析源代码,从而得出变换后的下标值。如果我们在下标变换中进行并行处理,变换程序的可能执行路径数目将随线程数目成指数级增长。由于线程是占有 CPU 的最小单位,如果要求串行程序能够变换为多线程并行化代码,那么被处理的数据在并行处理时不应该存在数据依赖关系。例如由 i 变换为较复杂的形式 $(i+3)*5+i+2+3*4+((5*3)*i)+3$,即求下标变换表达式的值,如果在串行程序中,只需

对程序代码进行静态分析,易得变换后 i 为 $21*i+32$,如果把该变换表达式进行并行化,要通过静态分析出源代码要变换的值将大大增加了复杂度。为了便于说明问题,我们令当前的 $i=2$,于是问题转化为并行求解 $(2+3)*5+2+2+3*4+((5*3)*2)+3$,并且使得程序的求解过程不易被攻击者分析出来。

1.1 数据预处理

为了使并行处理的数据没有依赖性,我们提出数据预处理算法为:首先在不改变下标变换求值表达式语义的前提下,对表达式中所有操作数,用尽可能多的"()"操作符进行操作,直至"()"外仅剩余"+"或者"-",则表达式变为:((2+3)*5)+2+2+(3*4)+((5*3)*2)+3。然后按照操作数被嵌套"()"的重数进行分组,并把表达式分解为只有单个运算操作符连接的简单运算式子,且值未知的表达式用 M 替换,那么被"()"嵌套两次的运算式有 2+3、5*3,嵌套"()"一次式子的有 $M*5$、$3*4$、$M*2$。最后令 N 为所有简单运算式子中被括号嵌套最大的重数,用 $N+1$ 个 List 或者数组分别存储被嵌套不同重数"()"的运算式子中的操作数。数据存储分布如图 1 所示,数据依赖关系用实线箭头标出,其中 M 可以存储任意值,用以迷惑攻击者。从图 1 可知同一个数组或者链表中的数据之间没有依赖关系,因此可以多线程并行处理。

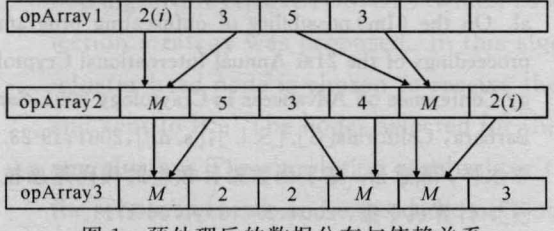

图 1 预处理后的数据分布与依赖关系

1.2 多线程并行执行的控制

为了隐藏下标变换过程,则要使得下标变换程序的执行路径应该具有随机性,同时要保证变换结果的正确性,因此我们必须对线程加以控制。并行控制的基本思想为:通过设置变量来保证程序运行同步,即只有在处理被依赖的数据的操作执行完成后,才能处理其他数据,而没有依赖关系的数据可以并行处理。

在本实例中,如下所示的控制线程代码就是通过设置数组变量"cls[]"来确保依次先后处理有数据依赖关系的 opArray1、opArray2、opArray3,其中的部分源代码只给出了保证同步处理 opArray1 中的条件"cls[0]< 4"。

```
if(cls[0]<4){
taskLock. tryLock();
try{
if(c[1]== 0 &&. c[2]==0)
{
    c[1]++;
    c[2]++;
    cls[0]=cls[0]+2;
    objectnum. twoand3();
}
}
finally{
    taskLock. unlock();
}
taskLock2. tryLock();
try{
if(c[3]==0 &&. c[4]==0)
{
    c[3]++;
    objectnum. fivemu3();
    c[4]++;
    cls[0]=cls[0]+2;
}
}
finally{
    taskLock2. unlock();
}
}
```

为了使得程序可能执行路径数目为无限大,我们在实现的时候,对每个线程使用了无限循环,根据需要也可以把线程设置一段时间后自动退出,由于一个线程的无限循环,我们必须设置一个变量数组"c[]"来保证每个操作数只被执行一次,这就确保了不同的线程可以并行的执行同一个 opArray,并且每个操作数只参与一次运算。同时为了防止在多核机器上多个线程同时访问同一个变量的情况发生,我们利用 java 中的锁机制实现,程序执行流程如图 2 所示。

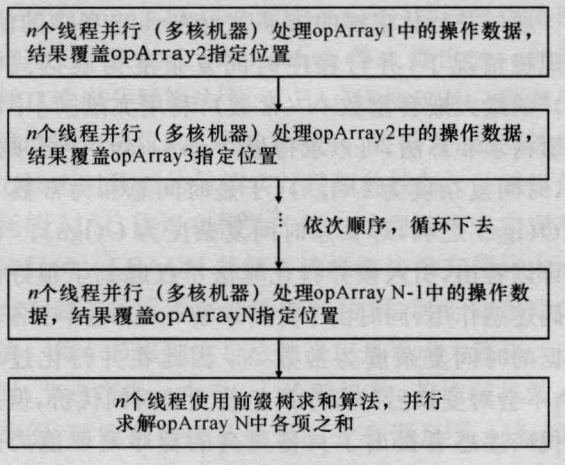

图 2 变换算法的执行流程

2 算法分析

基于Java多线程的隐藏数组下标变换表达式的代码迷惑算法是从代码迷惑的角度考虑，它可以用于各种基于控制流变换的代码迷惑技术来隐藏下标变换程序的执行路径。该算法提出的通过数据预处理来消除数据依赖性的思想，可以用于其他运算表达式的并行求值运算。用该并行化算法处理的迷惑代码，拥有很好的抗击源代码静态分析和基于源代码植入反迷惑攻击[6]的优点。如客户端线程启动代码为

```
public class MultiThread {
public static void main(String[] args)
        throws InterruptedException {
  ObjectNum objecttest = new ObjectNum
  ();
  Task task = new Task(objecttest);
  for (int k = 0; k < 5; k++){ //k 为线程
  数目
    new Thread(task). start();
  }
  Thread. sleep(100);
  for (int i = 0; i < 6; i++){
  System. out. println("outopArray3:"+
  objecttest. opArray3. get(i). toString());
  }
  }
  }
```

可以看出，客户端只是启动了若干线程，线程执行的顺序是随机的，而且可能执行路径随线程数目成指数级增长来决定；攻击者不能通过植入源代码来获取与程序执行顺序相关的信息，因为植入的源代码对所有线程可见，那么打印出的数据必是随机线程产生与程序执行顺序无关的信息，因此并行化的程序具有比较理想的迷惑效果。

通过并行化求解的结果为opArray3各项的值。在理想情况下，并行程序时间复杂度为表达式中“()”的最大嵌套重数 N（常数），利用文献[7]中的前缀树求和算法，可以求得最终opArray3中各项之和，时间复杂度为 $O(\lg n)$，于是时间总和为常数 N 与 $O(\lg n)$ 之和，那么总时间复杂度为 $O(\lg n)$。由此可以看出，引入该并行化算法不仅起到了很好的代码迷惑作用，同时由于并行求解opArray3中各项数据的时间复杂度为常数 N，因此在并行化过程中，不会对变换后的程序产生较大的时间代价，但是对代码迷惑者提出了具备更高的程序驾驭能力的要求。

3 结束语

随着多核时代的到来，并行计算的研究热潮必将促进并行化迷惑技术的进一步发展。为了防止攻击者分析出操作数据之间的逻辑关系，我们将进一步优化本文提出的算法，使得数据的存储位置不暴露其逻辑关系。例如，合并opArray1、opArray2、opArray3中的数据，都存储在opArray1中，并通过操作数组或者链表opArray1的下标来保证程序的正常运行。同样把各种控制变量也合并到一个数组或链表中，以掩盖不同控制变量之间的控制信息，而这些控制信息只有实施迷惑者才知道。为了防止攻击者通过单步调试程序来分析源代码，我们还将考虑利用更为复杂的加锁功能，即给锁设置一个超时属性值，一旦程序单步运行，程序将抛出超时异常。

参考文献：

[1] Collerberg Christian, Clark Thomborson, Doug Laslow. A taxonomy of obfuscating transformations [R]. New Zealan: Department of Computer Science, the University of Auckland, 1997: 148.

[2] Smiths. Secure coprocessing applications and research issues, LAUR-96-2805[R]. Los Alamos: Los Alamos National Laboratory, 1996.

[3] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, et al. On the (Im) possibility of obfuscating programs: proceedings of the 21st Annual International Cryptology Conference on Advancess in Craytology 2001, Santa Barbara, California[C]. [S. l.]: [s. n.], 2001: 19-23.

[4] 李永祥，陈意云. 基于函数指针数组的代码迷惑技术[J]. 计算机学报，2004, 27(12): 1706-1711.

[5] Zhang Xuesong, He Fengling, Zuo Wanli. An inter-classes obfuscation method for Java program: proceedings of the 2008 International Conference on Information Security and Assurance, April 24-26, 2008, Busan, Korea[C]. [S. l.]: [s. n.], 2008: 360-365.

[6] 卿海军，钟诚，张莲. 基于源代码植入的针对函数指针数组的反代码迷惑：2008计算机技术与应用进展[C]. 合肥：中国科学技术大学出版社，2008: 1095-1099.

[7] 陈国良. 并行算法的设计与分析[M]. 第3版. 北京：高等教育出版社，2009.

（责任编辑：韦廷宗）