第 32 卷 第 20 期
2010 年 10 月

武 汉 理 工 大 学 学 报
JOURNAL OF WUHAN UNIVERSITY OF TECHNOLOGY

Vol .32 No .20
Oct . 2010

DOI :10 .3963 /j .issn .1671-4431 .2010 .20 .036

可信计算动态验证优化建模

余 跃[1],余发江[1,2],孔亚楠[1]

(1 .武汉大学计算机学院 ,武汉 430072 ;2 . 武汉大学空天信息安全与可信计算教育部重点实验室 ,武汉 430072)

摘 要: 目前的可信计算平台只验证应用程序的静态散列值 ,不能防止恶意代码对应用程序的动态攻击。提出了一个基于静态分析的可信计算动态验证行为建模方法 ,并且设计了一种修改后的 Floyd 查找算法 ,用于找出并删除 FSA 中所有的空循环路径 ,优化模型。经过实验证明 ,算法无论是在时间效率上 ,还是在空间复杂度上都优于目前常用的基于图论的算法。

关键词: 可信计算; 动态验证; 行为建模; 空循环路径; 优化

中图分类号: TP 301 .4 **文献标识码**: A **文章编号** :1671-4431(2010)20-0169-05

Optimizing Behavior Model Building for Trusted Computing Dynamic Verification

YU Yue[1] , *YU Fa-jiang*[1,2] , *KONG Ya-nan*[1]

(1 .School of Computer , Wuhan University , Wuhan 430072 , China ; 2 . State Key Laboratory of Aerospace Information Security and Trusted Computing , Ministry of Education in China ,Wuhan 430072 , China)

Abstract: Current trusted computing platform only verifies the static Hash value of applications , which could not prevent the applications from being dynamically attacked by malicious code . This paper gives one static analysis-based behavior model building method for trusted computing dynamic verification . To optimize our model , we give an improved Floyd algorithm to find and remove ε run cycles in FSA . According to experiment , our method is much superior to the widely used algorithms based on graph theory both in time complexity and space complexity .

Key words: trusted computing ; dynamic verification ; behavior model ; optimizie ; algorithm

可信计算是针对计算系统的基础安全问题而提出的一种解决方案 ,是目前信息安全领域的研究热点之一[1,2]。但是 ,现有可信计算平台提供的完整性报告中 ,仅仅包括对应用程序运行前的静态完整性测量 ,在程序运行之后 ,仍然有可能被注入恶意代码 ,产生异常行为。所以 ,为规避或减少由于异常行为所导致的安全风险 ,还需要对程序动态行为进行验证。

目前 ,针对可信计算平台动态完整性的研究还比较较少 ,相关工作主要集中在基于系统调用的异常行为入侵检测领域 ,这类工作最为关键的是对程序的系统调用信息进行建模。Wagner[3]等提出了通过静态分析

收稿日期 :2010-06-20 .
基金项目 :国家自然科学基金(60673071 ,60970115),国家 863 项目(2006AA01Z442 ,2007AA01Z411),教育部空天信息安全与可信计算重点实验室开放基金(AISTC2008Q03)和浙江省信息安全重点实验室开放基金 .
作者简介 :余 跃(1988-),男 ,本科生 .E-mail :yuyue-whu@ foxmail .com

代码来建立行为模型的方法。静态分析目前采用的方法主要是建立程序行为的不确定有穷自动机(Nondeterministic Finite Automaton, NFA)和下推自动机(Push Down Automaton, PDA)模型。NFA 模型比较高效,但存在不可能路径,攻击者可能利用这些路径躲避检测。PDA 模型去除了 NFA 模型中的不可能路径,但在函数调用嵌套过多,栈所占用的空间会很大,运行代价较高[4-6]。

目前的相关工作,主要针对 Linux、UNIX 类操作系统,而且对软件进行行为建模时效率不高。在建模过程中,由于一些 FSA 中存在大量的空循环路径,可能在使用 FSA 模型匹配软件行为序列时,自动机陷入死循环无法停止。然而,在现有的行为建模方法中,有些方法没有考虑这个问题,有些方法采用的算法存在较为明显的问题。

作者提出了一个 Windows 平台下,基于静态分析的可信计算动态验证行为建模方法,并且设计了一种基于 Floyd 的查找算法,用于找出并删除 FSA 中所有的空循环路径。经过实验证明,算法无论是在时间效率上,还是在空间复杂度上都优于目前常用的算法。

1 可信计算动态验证行为建模

可信计算动态验证的整体模型如图 1 所示,与 Linux 等开源环境不同,在 Windows 环境下,无法得到许多重要的程序源代码,只能得到程序的二进制代码。所以,采用静态分析的方法进行行为建模。

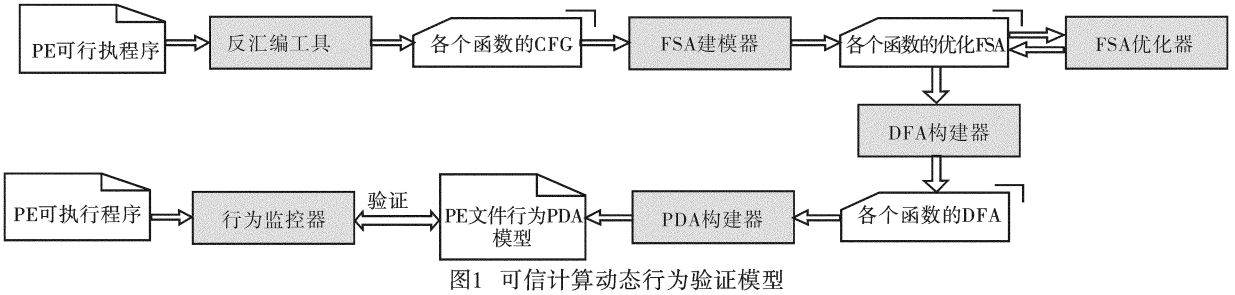

图1 可信计算动态行为验证模型

采用基于静态分析的方法进行建模,主要分为以下 6 个步骤:

1)从二进制代码生成 CFG:采用 IDA 的"wingraph32"插件,生成 PE 文件每个子函数对应的 CFG 图。CFG 表示为 $G=<V,E>$,V 是一个有限集合,V 中的元素 $v\in V$,为线性指令序列中的节点。E 是 $E(V)$ 的一个子集,$E(V)=\{(u,v)\mid u,v\in V\}$,$E\subseteq E(V)$,$E$ 中的元素是节点之间的边。

2)由 CFG 构建 FSA:只关心 Win32 API 和其他的子函数调用,所以,在 CFG 建立完后,应该过滤部分顶点上的无用指令,如 PUSH、JMP 等。

3)消除 FSA 的 ε 转换:在构建完每一个子函数 FSA 后,这些 FSA 中存在许多 ε 转换。这些 ε 转化的存在,不仅会浪费验证的时间,还可能导致 FSA 陷入死循环,所以必须被消除。这部分内容是优化建模的关键,该文将在第二章详细阐述。

4)构建确定 FSA:如果一个 FSA $M=(Q,\sum,\delta,S,F)$有多个初始状态,或者一个状态和输入符号存在不只一个后续状态,那么这个 FSA 是不确定 FSA(NFA),否则为确定 FSA(DFA)。在 NFA 中,可能存在多个转换:$\delta(q_i,a)=q_j\in\delta,\delta(q_i,a)=q_k\in\delta,q_j\neq q_k$。

如果直接用 NFA 做动态验证,同一个 Win32 API 序列需要更多的尝试。由于一个 NFA,必然存在一个等价的 DFA $M'=(Q',\sum',\delta',q_0',F')$。所以,采用子集构造算法将 NFA 转化为 DFA。

5)无用 FSA 的消除:首先,给出以下定义。

定义 1 存在一个 FSA $M=(Q,\sum,\delta,S,F)$。如果

(1)只存在一个顶点 $Q=\{q\}$为初始状态,同时也是最终状态;

(2)变换函数集 $\delta=\phi$;那么 M 是一个无用 FSA。

上述定义的无用 FSA 不存在任何转换,它对 Win32 API 序列的动态验证毫无作用,去掉它们可以精简构建的模型。

6)构建全局 PDA:动态验证是对应一个完整的应用程序,所以必须根据所有的局部子函数 DFA,构建出一个全局 DFA。由于一个子函数可能在多个地方被调用,所以子函数必须返回到正确的位置,全局 DFA 必须是全局 PDA,否则全局 DFA 就会存在"不可能路径"。

至此,就可以对一个 Windows 平台上的二进制文件,构建出的全局 PDA 模型,这个模型就是可信计算动态验证的行为模型。

2　查找 FSA 空循环路径

由上一节可知,利用该文的方法进行建模时,由 CFG 构建出的 FSA 模型里存在大量的空循环路径。如何消除这些 ε空循环路径,直接关系到建模方法的优劣。

1)基于图论的查找方法　现有的技术在建模时,通常使用基于图论的方法去寻找 FSA 中的空循环路径,如王玉英[10]等提出的"一种生成有向图全部简单回路的优化算法";徐兵[11]等提出的"有向圈的矩阵算法及有关性质"等。这一类算法的主要思想很类似,主要是对图中顶点进行缩减,在缩减过程中利用字符串标记保存图中原有信息,不断减少图中顶点的数量,最终将图缩为一点,逐步得到全部简单回路。

2)基于 Floyd 最短路径的查找方法　基于图论的算法虽然实现比较简单,但 FSA 中的有些节点肯定不会包含在任何一条循环路径中,而这种方法在没有对这些节点进行精简。不仅如此,基于图论的方法每一步都需要保存上一轮迭代计算出的路径,会占用较多的内存资源,这点将会在下一节中详细阐述。针对这些问题,采用一种修改后的 Floyd 算法,来查找 FSA 中的空循环路径。

定义 2　如果一个 FSA $M=(Q,\sum,\delta,S,F)$。M 存在一条路径,$u=a_0a_1\cdots a_{n-1}$($a_i\in\sum$,$i=0,1,\cdots,n-1$),这里的路径是一条状态序列,$q_0,q_1,\cdots,q_n$($q_i,q_{i+1}\in Q$,$\delta(q_i,a_i)=q_{i+1}$,$i=0,1,0,n-1$),标记为 R。

如果 $a_i=\varepsilon$,$i=0,1,\cdots,n-1$,则 R 是一条 ε路径;如果 $a_i=\varepsilon$,$i=0,1,\cdots,n-1$,其中 $q_i\neq q_j$,$i\neq j$,除了 $q_0=q_n$,则 R 是一条 ε空循环路径。

首先,要确定一个 FSA 中是否存在空循环路径,所以可以先用拓扑排序对 FSA 进行处理,剩下的 FSA 必然存在空循环路径。然后,为了提高建模效率,不需要找出全部的循环路径,而是找出其中最短的一条。因为,删掉一条路径上的相关节点,其余的循环路径就可能消失了。

```
For i from 0 to ε run state Number
    For j from 0 to ε run state Number
        LoopPath(i, j)= -1
        If there has δ(q_i,ε)= q_j ∈ δ then Dist(i, j)=1
        Else Dist(i, j)= ε run staten umber+1
For k from 0 to ε run state Number {
    For i from 0 to ε run state Number
        For j from 0 to ε run state Number
            If Dist(i, j)> Dist(i,k)+ Dist(k, j) then {
                Dist(i, j)= Dist(i,k)+ Dist(k, j)
                LoopPath(i, j)= k
            }
}
```

图2　修改后的Floyd算法

```
MinCycleLen = ε run state Number + 1
For i from 0 to ε run state Number
  If( Dist(i,i)≠ε run staten umber+1)
    {
      If( Dist(i,i)≤ MinCycleLen)
        MinCycleLen = Dist(i,i)
      Node = i;
    }
If(MinCycleLen ≠ ε run state Number + 1)
{
  Findpath(LoopPath , Node, Node );
}

Findpath( LoopPath , I , j )
{
  Temp= LoopPath( i , j );
  If( Temp = -1 )
    Return;
  FindPath( LoopPath , i , Temp );
  记录 Temp 中间节点;
  FindPath( LoopPath , Temp , j );
}
```

图3　求解最短循环路径的长度与路径上的节点

接着,用修改后的 Floyd 算法(如图 2 所示),计算每个节点的之间的最短路径并保存下来。计算节点间的最短路径后,利用算法 4(如图 3 所示)可以找出这些节点中最短的一条循环路径的长度,并递归求解出这条路径上的每一个节点。

3　算法分析和对比

1)时间复杂度　从上一章节的叙述可知,设计的算法可以细化为以下 3 个部分:(1)拓扑排序,确定是否存在空循环路径;(2) Floyd 算法,计算每个节点之间的最短路径;(3) 递归查询算法,查询最短的一条循环路径上个各个节点。

定义 3 $T=(n,n',e)$,用于表示时间消耗。其中,n 为 FSA 中节点数目,n' 为 FSA 空循环包含的节点数,e 表示 FSA 有向边的数量。定义 m_i g_i f_i,其中 m_i 表示减少的 FSA 节点数目,g_i 表示减少的空循环节点数,f_i 表示减少的边数。

可以计算修改后的 Floyd 算法查找出所有空循环路径所需要时间为

$$T=\sum_{i=0}^{k}[(n-m_i)(e-f_i)+(n'-g_i)^3+(n'-g_i)]\ (m_0,f_0,g_0=0)$$

经过展开计算可得

$$T=k(ne+n^{\beta}+n')\sum_{i=0}^{k}[(m_i f_i-e f_i-n f_i)+(3n g_i^2-3n^{\beta} g_i-g_i^3-g_i)$$

所以,当 n' 都非常大的时候,也就是构成空循环路径的节点数很多时,$T\to kn^{\beta}$,k 为一个常数。由此可见,算法时间主要集中在 Floyd 算法计算节点间的最短路径上,即时间复杂度为 $O(n^3)$。

与此算法相比,传统算法没有经过拓扑排序这一步,所以必然浪费大量的时间在一些无用节点之间的路径计算上。而且,基于图论的算法在这一点上是 NP 问题[10],算法复杂度是指数级的,所以,这类算法的时间复杂度为:$O(n^3 2^{2n})$。不仅如此,这类算法需要计算出全部的循环路径才能停止,中间路径存储涉及到字符串连接等操作同样需要消耗一定的时间,而且存储的中间路径越多、路径越长,所需要的时间也越长。

所以,总体来看,在时间效率方面,该算法比较占优,具体优势体现为以下 3 点:1)拓扑排序后,减少大量无用节点之间的运算;2)只计算出一条最短的循环路径路径,减少了删除的时间;3)只需要简单递归查找出确定的最短循环路径上的节点,无字符串连接等冗余操作。

2)空间复杂度 在空间占用上,传统算法有着致命的缺陷。这类方法在每一轮迭代计算时,需都要保存上次产生的中间路径,如果构成循环路径的节点过多,有向图节点间的路径过于复杂,则有可能由于存储了过多的中间路径,从而消耗大量的内存资源,并随着迭代次数而成倍增长。当存储量到达一定值时,就会发生内存泄露,导致系统崩溃。

与之相反,该算法只需要 2 个矩阵,一个矩阵用于存储各个节点之间最短路径的长度,另一个用于存储相邻节点的关系。

4 实验与分析

为了比较实际中的时间效率和空间消耗,选取了 Gdi32 .dll、Rpcrt4 .dll、User32 .dll 中具有代表性的 8 个子函数进行测试。

1)时间效率测试 分别使用文中的算法和传统的算法对上述文件进行处理,为了保证 2 类算法在寻找空循环路径时,处理节点数相同的样本,在传统算法中同样加入拓扑排序这一步,具体测试结果,如表 1 所示,其中算法 1 为该设计的算法,算法 2 为修改后基于图论的查找算法。

表 1 测试结果

子函数名称	FSA 节点数目/个	空循环的节点数目/个	找出空循环路径数目/条	用时时间/s	
				算法 1	算法 2
tree-size-ndr	48	45	20	6 .733 020	7 .076 274
tree-into-ndr	51	48	22	9 .021 329	8 .502 247
data-into-ndr	54	51	24	10 .370 068	10 .337 455
sub-77D5FAD4	82	81	31	33 .119 412	32 .868 784
data-from-ndr	104	85	33	36 .862 505	36 .543 700
sub-77D3B1DA	157	105	4	17 .867 722	内存溢出
sub-77C481F5	179	101	41	61 .411 450	64 .731 101
sub-77D3B6AF	177	153	60	221 .496 020	229 .541 671

从上述测试结果可知,当空循环节点数量较少时,此算法与基于图论的查找算法相差甚微。但随着节点

数量增大,相互关系逐渐复杂,此算法远远优于的基于图论的查找算法。

2)空间消耗测试　在空间复杂度方面,统计了 2 种算法在处理样本时,消耗内存资源的情况。图 4 根据时间点,列出 2 个算法正常处理测试样本时的资源消耗情况,图 5 则显示出使用基于图论的查找算法,在处理“sub-77D3B1DA”样本的 FSA 时发生溢出错误的资源消耗情况。

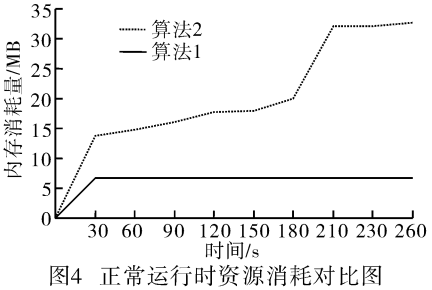

图4　正常运行时资源消耗对比图

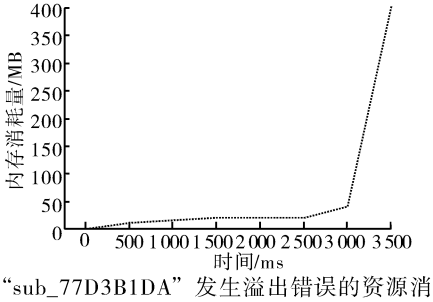

图5　“sub_77D3B1DA”发生溢出错误的资源消耗图

5　结　语

提出了一种 Windows 环境下,通过对二进制文件进行静态分析,为可信计算动态验证建立模型的方法。还提出了一种基于 Floyd 的改进查找算法,用于找出和消除各个局部 FSA 中的空循环路径,这一步骤大大优化了所建立的模型,是建模的关键所在。从实验结果来看,建模方法无论是从实际的时间效率方面,还是在空间资源消耗方面,都优于现在通用的算法。

参考文献

[1]　沈昌祥,张焕国,王怀民,等. 可信计算的研究和发展[J].中国科学:信息科学,2010,40(2):139-166.

[2]　沈昌祥,张焕国,冯登国,等. 信息安全综述[J]. 中国科学 E 辑:信息科学,2007,37(2):1-22.

[3]　Wagner David, Dean Drew. Intrusion Detection Via Static Analysis[C]//Proceedings of 2001 IEEE Symposium on Security and Privacy, Oakland: IEEE Computer Society, 2001: 156-168.

[4]　Giffin J, Jha S, Miller B. Detecting Manipulated Remote Call Streams[C]//In: Proceedings of the 11th USENIX Security Symposium, San Francisco, 2002: 61-79.

[5]　Feng H H, Giffin J, Huang Y, et al. Formalizing Sensitivity in Static Analysis for Intrusion Detection[C]// Proceedings of IEEE Symposium on Security and Privacy, IEEE Press, 2004: 194-208.

[6]　Gopalakrishna R, Spafford E, Vitek J. Efficient Intrusion Detection Using Automaton Inlining[C]//Proceedings of 2005 IEEE Symposium on Security and Privacy, IEEE Press, 2005: 18-31.

[7]　王玉英,陈　平,苏　旸. 生成有向图中全部简单回路的一种有效算法[J]. 计算机应用于软件,2009,26(12):27-29,33.

[8]　徐　兵,贾仁安. 有向圈的矩阵算法及有关性质[J].南昌大学学报:自然科学版,2002,26(1):5-11.