http://qks.cqu.edu.cn

文章编号:1000-582X(2012)02-065-06

恶意代码的符号执行树分析方法

钟金鑫,魏更宇,安 靖,杨义先
(北京邮电大学 信息安全中心,北京 100876)

摘 要:在恶意代码分析中,动态监测虚拟环境中的恶意代码行为是一种常用的方法。但是,由于可执行的路径分支众多,极易产生路径爆炸问题,造成某些可执行路径无法被覆盖,严重影响分析的全面性。为了解决恶意代码分析中路径爆炸问题,提出了一种基于符号执行树的恶意代码分析方法。通过构造符号执行树,引入汇聚节点,对恶意代码的执行路径进行约束求解,减少分析路径,从而缓解路径爆炸的影响,提高分析的全面性。恶意代码样本分析的实验表明,该方法能够有效地提升分析效率,同时拥有较小的时间复杂度。

关键词:符号执行;路径爆炸;恶意代码分析;汇聚节点;二进制程序分析
中图分类号:TP393 **文献标志码**:A

A malware analysis method based on symbolic execution tree

ZHONG Jin-xin, *WEI Geng-yu*, *AN Jing*, *YANG Yi-xian*
(Information Security Center, Beijing University of Posts and Telecommunications, Beijing 100876, P. R. China)

Abstract: In the malware analysis, it is a common method to monitor malware dynamically in a virtual environment. However, with so many branches of executable pathes, path explosion problem will probably occur, leaving some executable pathes uncovered, and hence harming the comprehensiveness of analysis. To solve this problem, we propose a malware analysis method based on symbolic execution tree. This method introduces sinknode and solves the execution of malicious code path by constructing the symbolic execution tree, so improves the analysis of comprehensive. Experiments to analyze the samples of malware show that the method can enhance the efficiency of the analysis with lower time complexity.

Key words: symbolic execution; path explosion; malware analysis; sink node; binary analysis

随着恶意代码的编写、传播以及利用发展成为一个的商业产业链[1],它已经严重地威胁到网络安全。为了能够及时有效地对恶意代码进行检测,需要对其进行样本分析,了解其程序运行机理及触发机制[2],从而提取恶意代码的特征码[3-4]及行为模式[5]。

现在对恶意代码的分析方法主要采用动态分析方法[6-8]。该方法将恶意代码样本在虚拟机等特定环境中进行培育运行,通过监控样本与虚拟机的交互行为,分析这些行为的函数调用关系和关键系统调用,从而获得恶意代码的运行机理以便查杀,如CWSandbox[9],NormanSandbox[10]以及 Anubis[11]等。但是,由于恶意软件在进行二进制程序分析时产生大量可执行路径分支,加之混淆及变形技术的使用,使得程序分支数骤增,导致路径爆炸[12-16],造成在可容忍的时间内无法完成全路径分析,从而不能够完全覆盖到恶意行为所有的触发情况,影响恶意代码分析的全面性。

收稿日期:2011-06-15
基金项目:国家自然科学基金资助项目(61070208)
作者简介:钟金鑫(1984-),男,北京邮电大学博士生,主要从事信息安全方向研究,(Tel)13811653449;
(E-mail)jinxin.zhong@gmail.com。

http://qks.cqu.edu.cn

尽管,有典型的解决方法被提出,但都没有取得很好的效果。Andreas Moser 等人[12]提出了一种基于多执行路径的恶意代码分析方法,由于构造了所有可能的执行路径,使得局部路径出现大量冗余数据,造成资源大量消耗的同时,也影响了分析的效率,使得恶意代码分析的路径覆盖率不足 85%。王祥根等人[17]提出的基于代码覆盖的多路径分析方法仅仅考虑了循环代码的处理,多路径分支及遍历都没有进行解决,从而无法进行全面分析。

笔者提出了一种基于符号执行树的恶意代码分析方法,该方法通过构造符号执行树,对恶意代码执行路径条件进行约束,使之形成新的汇聚节点,有效地避免了可达路径的重复遍历,减少了分支数,缓解了路径爆炸问题,提高了恶意代码分析的执行路径覆盖率。同时,基于符号执行树的分析方法在时间复杂度上也有了明显降低,提高了分析速度。实验表明基于符号执行树的恶意代码分析方法能够明显地缩短恶意代码分析时间,提高分析的效率及全面性。

1 符号执行树分析方法

基于符号执行树的恶意代码分析方法是一种基于前向分析的符号执行方法[18],它将符号作为程序输入,从前向后用符号依次替换中间变量,最终使程序各变量的值是关于符号输入的表达式或具体值。

1.1 概念定义

符号执行树分析方法是建立在程序控制流图的基础上的。它将二进制程序的结构表示为控制流图 $G=<N,E,\text{first},\text{end}>$,其中 N 表示执行树中的节点集合,表示程序中的基本块,E 表示图中边集合,若程序中有 n_i 跳转到 n_j 的语句,则在 G 中有一条从 n_i 到 n_j 的有向边,记为 $e=(n_i,n_j)\in E$,节点 first 和 end 表示表示控制流图的起点和终点。

为了更好地表示该方法中执控制流图的分析过程,定义的概念如下:

定义 1:环。在控制流图中存在回边 $e=(n_j,n_i)$,且从控制流图起点到 n_j 的所有路径必然经过 n_i,即 n_j 是 n_i 的必经点,则用回边 $e=(n_j,n_i)$ 标识一个自然环,该环由 n_i 和所有不经过 n_i 可到达 n_j 的节点集合以及连接这些节点的边所组成的子图。

定义 2:执行树。执行树是在控制流图的基础上转换而来的,表示为 $\text{ExeTree}=<N,E,\text{root},\text{leaf}>$,其中 root 节点为树的根节点,leaf 是叶子节点(表示正常退出或者异常退出),执行树中的每个节点只包含一个父节点,每一条执行路径是由顺序执行的各个节点组成。

$$P=\{n_i \mid n_i\in N, i=0,1,\cdots,m\}, \tag{1}$$

其中:m 表示路径的长度;n_i 是 n_{i+1} 的父节点;n_0 是根节点;n_m 是叶子节点。

定义 3:原子路径。将相邻两节点 n_{i-1},n_i 之间的路径分支 $e_i=(n_{i-1},n_i)\in E$ 称为原子路径。将从原子路径的父节点 n_{i-1} 执行到子节点 n_i 的路径条件称为原子路径条件,用 C_i 表示。

定义 4:节点约束条件。节点约束条件为从根节点执行到该节点的路径条件,对于节点 n_i,路径约束条件 C_{Pi} 为

$$C_{Pi}=\bigwedge_{N_i\in P} C_i, \tag{2}$$

其中 C_i 表示从根节点到节点 n_i 的执行路径上的各原子路径条件。

定义 5:汇聚节点。汇聚节点是在节点约束条件下符号执行路径不断汇聚形成的新节点

$$C_{Pi}=C_i \wedge C_{Pi-1}, \tag{3}$$

其中:C_i 为原子约束条件;C_{Pi-1};C_{Pi} 分别表示节点 n_{i-1},n_i 的约束条件。

当汇聚节点 n_i 为叶子节点,表示该执行路径结束,此时叶子节点的约束条件即为求解的某一个路径条件。一棵完整的执行树结构模型如图 1 所示。

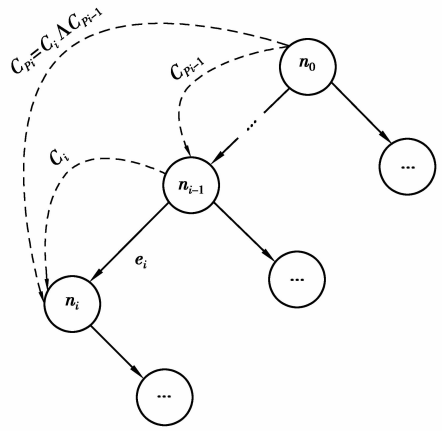

图 1 执行树结构模型

1.2 环转换

在基于符号执行树的分析方法中,生成的符号执行树的每个节点应当只包含有一个父节点。在实际的程序代码中,这种情况几乎是不存在的。因为实际的程序不仅仅包含顺序分支结构,而且包含循环分支结构,这样控制流图 G 中不可能只包含一个父节点。在本方法中,为了保证满足此条件,需要对不符合条件的执行树进行环转换。

在控制流图 G 中,程序中的循环结构以有向环的形式存在,将这些环作为一个基本块,成为控制流

http://qks.cqu.edu.cn

图 G 的一个独立节点。

在某些条件下,当循环的条件的不断被满足,循环条件将不断被执行且一直执行下去,即程序中出现了死循环。这种循环结构的执行将会消耗大量的时间,因此大大降低程序分析的效率。

为了避免程序符号执行陷入死循环,为由环转换而来的节点设置一个阀值 Timeout,当该节点的执行时间超过阀值,终止该节点的执行,直接跳转到下一节点。如图 2 所示。

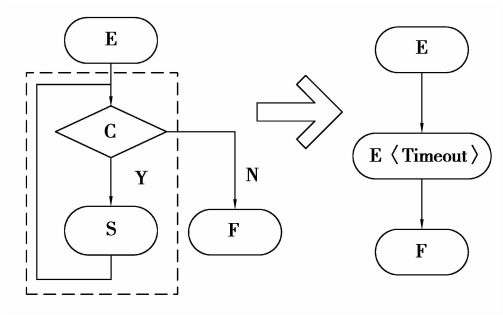

图 2　环转换示例图

1.3　条件分支结构转换

除了循环分支结构,控制流图 G 中还包含条件分支结构。如果控制流图 G 中包含有条件分支结构,当各条件分支执行完毕后执行跳转到同一个节点,那么该节点将包含多个父节点。为了保证父节点的唯一性,本方法对条件分支结构进行转换。

如果控制流图中的某个节点有 k 个父节点,则把这个节点分为 k 个节点,使每个节点只包含一个父节点。执行树中的根节点表示程序入口,箭头表示路径分支,每个叶子节点对应一条路径,如图 3 所示。

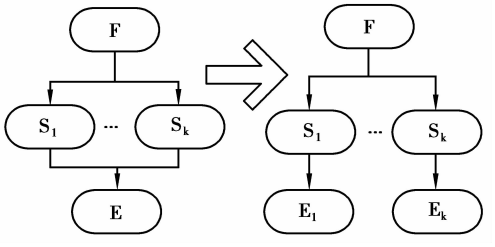

图 3　条件分支结构转换示例图

如果程序中存在大量分支结构,这种转换方式会大量增加节点的数量,但是不会增加可行的路径数,因此不会增加程序分析算法的复杂度。

2　算法设计

2.1　算法思路

所提出的基于符号执行树分析方法对程序进行分析的基本思路是:在路径执行过程中,当访问的节点是没有被访问过,则计算该节点的约束条件,并将该节点标记为已访问节点;当再次访问该节点时,不需要重新构造该节点的路径条件,直接跳转到该节点即可;当所有的节点被访问完毕时,即完成了对程序的分析。方法的基本步骤如下

1)利用广度优先遍历算法[19]得到未被访问的首个节点 start;

2)该 start 节点作为起始节点,利用基于符号执行树的算法进行分析,最终得到某条执行路径的执行条件,然后跳转到 1)执行。

研究方法不同于传统的遍历算法[18](广度优先或深度优先),在构造每条路径执行条件时,避免了重复构造原子路径条件,降低了算法复杂度。

2.2　算法实现

基于符号执行树的恶意代码分析方法的算法实现如图 4 所示。

```
1   Begin
2   Node start=root;
3   Cp=NULL
4   WHILE1(CurrentIsLast(start)==FALSE)
5     IF1 (visitTimes(start)!=1)
6       start = GotoNextNode(start);
7     ELSE1
8       n=start;
9       WHILE2 (CurrentIsLast(n)==FALSE)
10        IF2 (visitTimes(n)==1)
11          CopyFatherPC(Cp,n);
12          AddAtomPC(Cn,n);
13          Mark(n);
14          n= GotoChildNode(n);
15        ELSE2
16          n= GotoChildNode(n);
17        ENDIF2
18      END WHILE2
19      CopyNodetoPC(n,Cp);
20    ENDIF1
21  END WHILE1
22  End
```

图 4　恶意代码分析方法

1)Line 1～6:从 root 节点开始对符号执行树进行遍历,判断节点是否被访问过,如果没有,通过函

http://qks.cqu.edu.cn

数 GotoNextNode 跳转到广度优先遍历的下一个节点，直至找到未被访问过的首个节点 start。

2)Line 8-19：将 start 作为基于符号执行树分析算法的起始节点，判断遍历的节点是否是叶子节点，如果是，跳转到 Line 19；如果不是，判断该节点是否被访问过，如果否，跳转到 Line 11，反之，跳转到 Line 16。

3)Line 11-14：当节点未被访问过，首先调用函数 CopyFatherPC，将它的父节点的约束条件保存到该节点中，然后调用函数 AddAtomPC，将原子约束条件 Cn 添加到节点 n 的约束条件中，以获得子节点的约束条件，然后通过函数 Mark(n)将该节点标记为已遍历节点，然后继续遍历它的子节点。

4)Line 16：当节点已遍历过，则节点中已保存了路径约束条件，直接跳转到子节点。

5)Line 19：当遍历的目标节点是叶子节点时，提取该节点约束条件，即为某条可执行路径的约束条件。

2.3 算法实例分析

为了更好地说明基于符号执行树的分析方法的路径选择算法，通过实例来进行分析，图 5 是由控制流图转化而来的执行树实例。如果采用传统的基于深度优先遍历算法[19]（对应于树的先序遍历算法）。符号执行的路径依次为

1)1－>2－>4－>8－>11；

2)1－>2－>4－>8－>12；

3)1－>2－>5－>9－>13－>17…>n_1。

可以看出，对于基于深度遍历的算法，选择的下一条路径与当前路径有着最长相同前缀，因此前后 2 次的执行路径具有很大的相似性。如果节点 1 左子图存在路径爆炸问题，在有限的时间内不能访问到右子图，导致对恶意代码的分析不够全面。

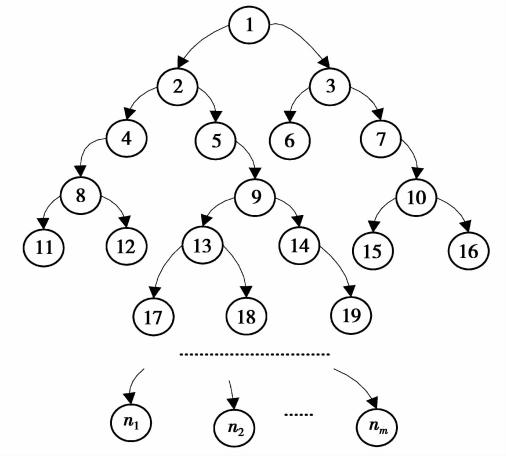

图 5 算法分析实例

采用基于符号执行树的分析方法对图 5 所示执行树进行遍历，从遍历的路径依次为

1)1－>2－>4－>8－>11；

2)3－>6；

3)4－>8－>12；

符号执行过程：算法从根节点（节点 1）开始进行符号执行，执行路径为 1)；在执行第二条路径时，首先通过广度优先遍历得到未被访问的首个节点（节点 3），从节点 3 开始进行符号执行，由于节点 3 没有被访问过，因此利用父节点约束条件和原子路径条件求解该节点的约束条件，继续访问它的子节点（节点 6），由于节点 6 是叶子节点，提取出节点 6 的路径条件即为 1－>3－>6 的路径条件。同理从节点 4 开始求解路径 4－>8－>12 的约束条件。

由此可得，该算法选择的下一条路径与当前路径有最短相同前缀。由于相同的节点越少，路径的相似性越小，扩散性越大，代码的覆盖率越大，从而缓解了路径爆炸导致的代码分析不全面的问题。

2.4 算法时间复杂度

为了量化所提出的基于符号执行树的分析方法时间复杂度，假定生成的执行树是一棵满二叉树，深度为 $h(h\geqslant 2)$。

采用传统的路径遍历算法[19]，无论是广度优先算法还是深度优先算法，当每次遍历同一个节点时，都必须计算节点对应的原子路径条件。对于一个深度为 h 的满二叉树，具有 2^{h-1} 条可行的路径，每条执行路径包含的原子路径数为 $h-1$，因此传统的路径遍历算法的时间复杂度为 $O((h-1)2^{h-1})$。

采用基于符号执行树的分析方法进行分析，对每一条原子路径只遍历一次，因此算法复杂度为 $O(2^{h-1}-1)$，即 $O(2^{h-1})$，与 $O((h-1)2^{h-1})$ 相比，明显降低了算法的时间复杂度。

3 实验结果和讨论

基于上述的分析方法，实现了一个基于符号执行树的恶意代码分析方法的原型工具 SEtool (symbolic execution tool)。由于现在大多数的恶意代码在 Windows 操作系统上传播和运行，因此 SEtool 的运行环境是 WindowsXP 操作系统，计算机硬件配置 CPU 为 Core i3 530，内存为 4 GB。

为了验证方法在恶意代码分析中的有效性和效率，设计了两组对比实验：1)分别使用基于多执行路径的恶意代码分析方法的二进制分析工具 TEMU[20] 和基于符号执行树的恶意代码分析方法

http://qks.cqu.edu.cn

的原型工具 SEtool,对 VirusBulletin 恶意代码样本测试包[21]中随机挑选的 300 个恶意代码样本进行分析,记录每个样本的分析完成时间,从而比较这两种方法的分析效率;2)使用 TEMU 与 SEtool 分别对 300 个恶意代码样本进行分析,记录每种样本的已执行路径数和总路径数,从而比较这 2 种方法的分析全面性。

3.1　分析效率比较

恶意代码分析效率[17]是指在单位时间内,完成分析的恶意代码样本数与总样本数的比值,即

$$\text{分析效率} = \frac{\text{完成分析样本数}}{\text{总样本数}} \times 100\%。$$

实验使用 TEMU 与 SEtool 分别对 300 个恶意代码样本进行分析,在 100 s 时间内,得到完成和未完成的分析样本数量如下表 1 所示。

表 1　两种工具的分析效率

分析工具	完成数	未完成数	分析效率
SEtool	287	13	95.7%
TEMU	169	131	56.3%

由对比实验可以看出,在相同时间内,基于符号执行树的恶意代码分析方法无论在分析完成的样本数量上还是在分析效率上都远远高于基于多执行路径的分析方法。随着恶意代码体积的增加,分支数将继续增加。

3.2　分析全面性比较

恶意代码分析全面性可以通过路径覆盖率来进行衡量。路径覆盖率是指在分析过程中,执行的路径数与总路径数的比值,即

$$\text{路径覆盖率} = \frac{\text{执行路径数}}{\text{总路径数}} \times 100\%。$$

本实验使用 TEMU 与 SEtool 分别对 300 个恶意代码样本进行分析,得到的路径覆盖率分布如下表 2 所示。

表 2　SEtool 与 TEMU 的路径覆盖率分布

路径覆盖率	样本数量		样本分布率	
	SEtool	TEMU	SEtool	TEMU
95%～100%	257	98	85.67%	32.67%
90%～95%	37	109	12.33%	36.33%
85%～90%	6	60	2%	20%
85%以下	0	33	0	11%

由表 2 的实验结果可以看出,使用了基于符号执行树的恶意代码分析方法的 SEtool 相较于 TEMU 在恶意代码分析全面性上有了比较明显的提高,所有分析样本的路径覆盖率都在 85%以上,85.67%的分析样本的路径覆盖率在 95%以上。TEMU 所分析的样本路径覆盖率在 95%以上的仅有 32.67%,并且存在有 11%的分析样本路径覆盖率在 85%以下。这是由于 TEMU 没有采用优化算法来对执行路径进行优化,导致执行路径重复,加之随着路径分支数的急剧增加,使之产生路径爆炸,从而在可容忍的范围内无法执行可达路径,从而降低了恶意代码分析的全面性。

4　结　论

提出了一种基于符号执行树的恶意代码分析方法,该方法在构造符号执行树中引入汇聚节点,对恶意代码执行路径条件进行约束,有效地避免了可达路径的重复遍历,缓解了路径爆炸问题。通过对比实验表明基于符号执行树的恶意代码分析方法能够有效地提高分析效率和路径覆盖率,从而提高恶意代码分析的全面性。

参考文献:

[1] Symantec. Internet security threat report, 2010 [EB/OL]. http://www.symantec.com/zh/cn/business/theme.jsp?themeid=threatreport.

[2] SONG D, BRUMLEY D, HENG Y, et al. BitBlaze: A New Approach to Computer Security via Binary Analysis[C]// In Proceedings of the 4th International Conference on Information Systems Security (ICISS). [S.L.]: IEEE, 2008,12: 28-32.

[3] MIHAI C, SOMESH J, SANJIT S, et al. Semantics-aware malware detection[C]// In Proceedings of the 2005 IEEE Security and Privacy Conference. [S.L.]: IEEE, 2005, 8: 87-92.

[4] JEFFREY W, CHIUEH T C. A forced sampled execution approach to kernel rootkit identification[C]// In Recent Advances in Intrusion Detection. [S.L.]: IEEE, 2007,9: 219-235.

[5] YIN H, SONG D, EGELE M, et al. Panorama: Capturing system-wide information flow for malware detection and analysis. [C]// In: Proc. of the 14th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2007: 116-127.

[6] MANUEL E, CHRISTOPHER K, ENGIN K. Dynamic spyware analysis [C]// In Proceedings of USENIX Annual Technical Conference. [S.L.]: IEEE, 2007,6:89-93.

http://qks.cqu.edu.cn

[7] VASUDERAN A, YERRABALLI R. Cobra: Fine-grained malware analysis using stealth localized-executions[C]// In SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06). Washington DC, USA: IEEE Computer Society, 2006: 264-279.

[8] GARFINKEL T, ROSENBLUM M. A virtual machine introspection based architecture for intrusion detection [C]// In Proceedings of Network and Distributed Systems Security Symposium (NDSS'03). [S. L.]: IEEE, 2003,2:128-132.

[9] WILLEMS C. CWSandbox: Automatic behaviour analysis of malware [EB/OL]. http://www.cwsandbox.org/.

[10] Norman ASA. Norman Sandbox [EB/OL]. http://sandbox.norman.no/.

[11] Anubis: Analyzing unknown binaries [EB/OL]. http://analysis.seclab.tuwien.ac.at/.

[12] ANDREAS M, CHRISTOPHER K, ENGIN K. Exploring multiple execution paths for malware analysis [C]// IEEE Symposium on Security and Privacy. DSA: IEEE Computer Society Press, 2007:231-245.

[13] 梅宏,王千祥,张路,等.软件分析技术进展[J].计算机学报, 2009,32(9): 87-92.
MEI HONG, WANG QIAN-XIANG, ZHANG LU, et al. Software Analysis: a road map[J]. Chinese Journal of Compaters, 2009, 32(9): 20-24.

[14] WANG T, WEI T, LIN Z, et al. Intscope: Automatically detecting integer overflow vulnerability in x86 binary using symbolic execution[C]// the 16th Network and Distributed System Security Symposium (NDSS09). San Diego, CA: IEEE, 2009, 2:156-160.

[15] ANAND S, GODEFROID P, TILLMANN N. Demand-driven compositional symbolic execution[C]// In Proceedings of Tools and Algorithms for the Construction and Analysis of Systems. [S. L.]: IEEE, 2008:182-187.

[16] BOONSTOPPEL P, CADAR C, ENGLER D R. RWset: attacking path explosion in constraint-based test generation[C]// 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, [S. L.]: IEEE, 2008. 2008: 351-366.

[17] 王祥根,司端锋,冯登国,等.基于代码覆盖的恶意代码多路径分析方法[J].电子学报, 2009,37(4):701-705.
WANG XIANG-GEN, SI DUAN-FENG, FENG DENG-GUO, et al. Exploring multiple excution paths for malware analysis based on coverage of code[J]. ACTA Electronica Sinica, 2009, 37(4): 701-705.

[18] JAMES K. Symbolic execution and program testing [J]. Communications of the ACM,1976, 19:386-394.

[19] THOMAS H, CORMEN, C E. Introduction to algorithms (second edition) [M]. Cambridge: MIT Press, 2001.

[20] TEMU: The BitBlaze dynamic analysis component [EB/OL]. http://bitblaze.cs.berkeley.edu/temu.html.

[21] The Wildlist-viruses out in the wild [EB/OL]. http://www.virusbtn.com/resources/wildlists/index.

(编辑 侯 湘)

(上接第 64 页)

[12] LAW C K. Combustion physics [D]. Cambridge University:Cambridge University Press, 2006.

[13] GLASSMAN I. Combustion, 3 rd edition [D]. San Diego, California: Academic Press, 1996.

[14] FIEWEGER K, BLUMENTHAL R, ADOMEIT G. Self-ignition of S. I. engine model fuels: a shock tube investigation at high pressure [J]. Combustion and Flame, 1997,109 (4) 599-619.

[15] HERZLER J, JERIG L, ROTH P. Shock tube study of the ignition of lean n-heptane/air mixtures at intermediate temperatures and high pressures [J]. Proceedings of the Combustion Institute, 2005, 30 (1): 1147-1153.

[16] DAVIDSON DF, GAUTHIER B M, HANSON RK. Shock tube ignition measurements of iso-octane/air and toluene/air at high pressures [J]. Proceedings of the Combustion Institute, 2005, 30 (1): 1175-1182.

[17] HERZLER J, FIKRI M, HITZBLECK K. Shock-tube study of the autoignition of n-heptane/toluene/air mixtures at intermediate temperatures and high pressures [J]. Combustion and Flame, 2007, 149 (1/2):25-31.

[18] GUSTAVSSON J, GOLOVITCHEV VI. Spray combustion simulation based on detailed chemistry approach for diesel fuel surrogate model[C]. SAE 2003-01-1848, 2003.

[19] FREDRIKSSON J, BERGMAN M, GOLOVITCHEV VI, et al. Modeling the effect of injection schedule change on free piston engine operation[C]. SAE 2006-01-0449, 2006.

[20] GOLOVITCHEV VI, CALIK AT, MONTORSI L. Analysis of combustion regimes in compression ignited engines using parametric φ-T dynamic maps[C]. SAE 2007-01-1838, 2007.

[21] GRIFFITHS JF. Reduced kinetic models and their application to practical combustion systems [J]. Progress in Energy and Combustion Science, 1995, 21 (1):25-107.

(编辑 陈移峰)