态.本文通过设置 *Range* 值来降低漏报率,通过加入 *IdleCountThreshold* 来降低误报率.在实际测试中,通常会多次测试被测程序,选取其中最大的 *IdleCountThreshold* 以及 *Range* 来进一步保证低漏报率与误报率.

图 11 中特征值图的情况如图 12 所示,图中所示程序 *IdleNGAver* 为 76,*Range* 值为 5,*IdleCountThreshold* 值为 2.

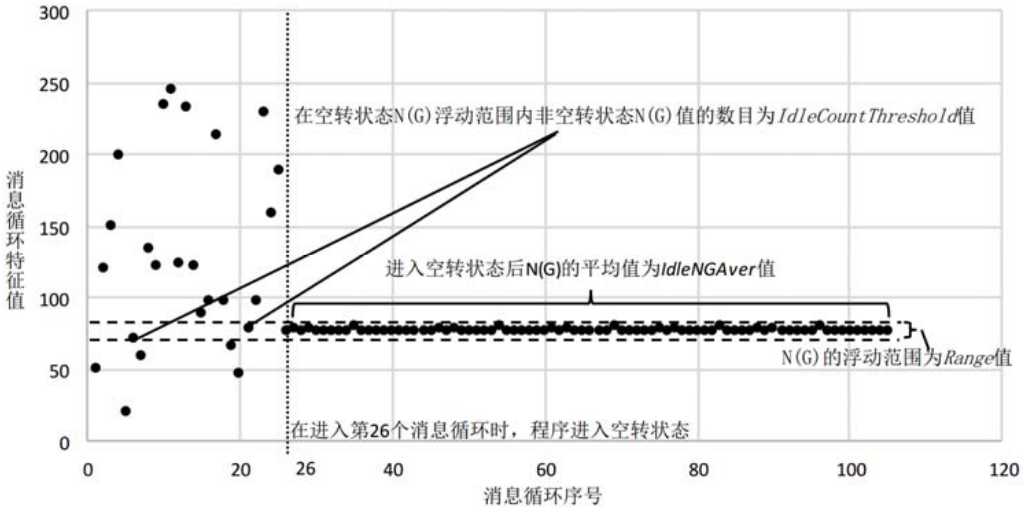


Fig.12 Method of idle state feature extraction in Section 5

图 12 利用第 5 节示例提取空转状态特征

5 空转状态的实时判定

空转状态实时检测就是在程序运行时,根据程序函数执行过程,实时生成 $N(G)$值,并根据 $N(G)$值是否进入空转状态进行实时判定.由于 $N(G)$值的计算仅仅涉及到整数加及内存查找运算,对被测程序运行效率几乎没有影响;$N(G)$值计算完成后,就需要判定程序是否进入空转状态.为了尽量降低判定过程对被测程序运行效率的影响,判断过程的时间复杂度一定要低.

由于 Bi-Gram 模型以及语料库的选择,程序的空转状态与非空转状态在 $N(G)$值上必然存在数值上的差异,因此,程序进入空转状态时,消息循环的 $N(G)$值必然在空转状态概率特征值范围内.基于此原理,本文提出的空转状态实时判定算法如下.

```
BOOL isInIdleState(double N(G))                    //函数参数 N(G)表示当前求得的消息循环的 N(G)值
{
  if (abs(N(G)-IdleNGAver)<Range){
    idleCount++;
    if (idleCount≥IdleCountThreshold){
      return True;}}                               //判定进入空转状态
  return False;}
```

对于消息循环执行迹来说,其函数个数 l 并不是个定值,这也就导致了可能会出现两个完全不同状态的消息循环的 $N(G)$值相同的情况;同时,由于程序在运行时可能会出现先加载界面再加载测试用例的情况,导致本应属于空转状态的 $N(G)$出现在非空转状态 $N(G)$集合里,因此在检测时,需要一个累计变量 *idleCount*,检测出现 $N(G)$的次数超过一定阈值后,判定程序进入空转状态.

在实时判定算法的基础上,进一步可以得到实时检测算法.算法在程序运行时利用语料库以及 3 个特征值根据程序实时函数执行迹判断程序是否进入空转状态.实时检测算法如图 13 所示.

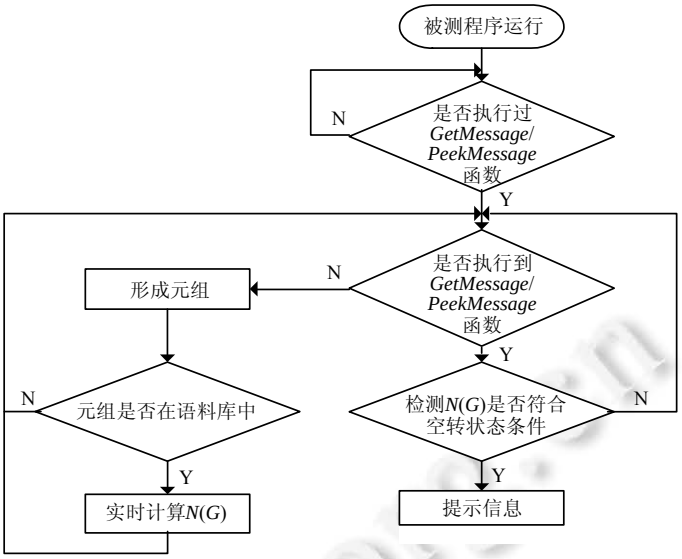


Fig.13 Flow chart of real time detection method

图 13 实时检测算法流程

由图13可知,实时检测算法在程序运行时实时构造元组数据,并采用实时检测算法判断当前$N(G)$是否满足空转状态条件.当算法检测到形成的元组不在语料库中时,说明该元组是不属于空转状态的,因此直接跳过元组生成等工作,等待下一个消息循环的进入,以此提高程序的执行效率.

6 实现与实验

本节实现了 Windows 下 GUI 程序空转状态实时检测原型工具,通过一系列实验验证了本文提出空转状态识别方法的准确性以及效率.首先,本文选用了修改过源码的带 GUI 程序 TestApp,以测试算法的准确性;然后,选择了 5 个真实存在的应用程序,通过模拟模糊测试的超时时限过程与使用算法进行比较,验证方法的效率.

6.1 工具实现

工具分为 3 部分,分别为函数执行迹记录部分、特征提取部分以及实时检测部分,整体关系如图 14 所示.

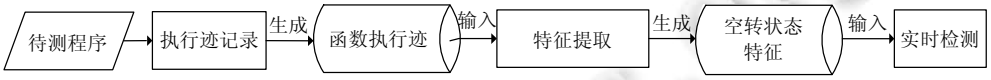


Fig.14 Relationship of every part of the tool

图 14 工具整体关系

对于 Windows 下带 GUI 界面的程序来说,程序的运行过程不仅与程序有关,也与相关的动态链接库有关.实际上,消息循环的主体函数大都是系统自带动态链接库的 API 函数,因此对程序的函数执行迹的记录,不仅仅需要记录程序自身内部的函数,对其所涉及的动态链接库的函数也要进行记录.由于本文分析的主要目标是程序本身的行为,对动态链接库相关的内部行为并不关心,因此对于动态链接库,函数执行迹的记录主要是针对动态链接库中的导出函数,而忽略其内部函数;对于程序本身来说,函数执行迹的记录主要是其内部自定义的函数.通过记录动态链接库的导出函数以及程序内部自定义函数执行轨迹来表示程序消息循环的特征.本文所需要记录的函数见表 1.

Table 1 Table of the needed function by the trace recorder

表 1 执行迹记录部分所需记录的函数

函数类型	是否需要记录
程序自身内部函数	√
动态链接库导出函数	√
动态链接库内部函数	×

本文采用 PIN[11]作为执行迹记录的平台,因此,动态链接库的导出函数可以直接通过符号加载器监控.然而,PIN 对程序自身定义的函数却没有办法进行监控,因此,本文首先使用 IDA PRO 对被测程序进行分析,生成自定义函数数据库.

模块在被测程序运行前,首先对程序自定义函数进行地址转换.由于在 Windows 7 或更高的系统中加入了地址随机化(ASLR),因此程序自身的加载基址会随着程序每一次执行而改变,因此,本文在程序加载进入内存中时,记录其加载基址,将自定义函数数据库内的地址进行转化,从而执行迹记录模块可以在程序运行时正确地找到程序自定义函数的入口.

特征提取模块主要有两个作用:一是生成及更新基于 Bi-Gram 模型的语料库,二是生成及更新空转状态判断特征.语料库提取功能首先读取函数执行迹数据库;然后以 *GetMessage*/*PeekMessage* 为分隔符,将函数执行迹分割为“句子”;最后,按照 Bi-Gram 模型提取出 $C(f_{i-1}\rightarrow f_i)$以及 $C(f_{i-1})$,并计算出 $\lg(P(f_i|f_{i-1}))$,将其保存在语料库中.

在语料库的更新过程中,语料库内主要保存 3 种数据:$C(f_{i-1}\rightarrow f_i)$,$C(f_{i-1})$以及 $\lg(P(f_i|f_{i-1}))$.假设原预料数据库内有 $C(f_{i-1}\rightarrow f_i)_{before}$,$C(f_{i-1})_{before}$ 以及 $\lg(P(f_i|f_{i-1})_{before})$,现加入一个新执行迹的数据,新执行迹提取出 $C(f_{i-1}\rightarrow f_i)_{now}$,$C(f_{i-1})_{now}$ 以及 $\lg(P(f_i|f_{i-1})_{now})$,则更新过后的语料数据库内有 $C(f_{i-1}\rightarrow f_i)_{new}=C(f_{i-1}\rightarrow f_i)_{now}+C(f_{i-1}\rightarrow f_i)_{before}$,$C(f_{i-1})_{new}=C(f_{i-1})_{now}+C(f_{i-1})_{before}$ 以及 $\lg(P(f_i|f_{i-1})_{new})=\lg(C(f_{i-1}\rightarrow f_i)_{new}/C(f_{i-1})_{new})$.通过这种方法完成语料库的更新.

针对多个 $N(T)$,本文选用对 *IdleNGAver* 取平均、对 *Range* 和 *IdleCountThreshold* 取最大值的方式实现空转状态特征值的更新.由于不同的 *IdleNGAver* 在同样的语料库下算得的差距并不会太大,因此采用取平均的方法;而 *Range* 表示属于空转状态特征的情况,为了尽可能地将满足空转特征的概率特征加入到判别阈值中,选用最大的 *Range*;同样,由于采用保守策略,尽可能地降低误报率,选取最大的 *IdleCountThreshold*.

实时检测模块在程序运行时利用语料库以及 3 个特征值实时判断程序是否进入空转状态.实时检测模块首先加载语料库以及自定义函数库,然后进行地址转换,最后开始验证工作.检测算法流程如图 13 所示.

6.2 实验与结果分析

本节介绍一个验证实验和一个对比实验.验证实验采用自行开发的的带 GUI 程序的 TestApp,TestAPP 实现了对文件的加载,并将文件的内容通过一定的运算,将运算结果显示到窗口上.该程序包含一个消息循环,文件处理过程采用单独的线程进行计算;同时,TestAPP 还在程序运行时统计执行过消息循环的个数,并在程序完成文件加载运算以及显示后停止计数,并将该计数作为结果输出.然后,使用空转状态识别算法对该程序进行空转状态识别,同样,记录程序被识别进入空转状态时所经历的消息循环的个数.通过对比两个消息循环的个数来判断算法的准确度.

对比实验选用 7 款商用软件,分别是 IKEView、WmDownLoader、VUPlayer、FoxitReader、WPS 表格、FlashFXP 以及 Dupsctc.对这 7 款软件采用空转状态识别算法,在识别到程序进入到空转状态时结束被测软件进程;同时,采用 PIN 加载这 7 款软件,分别统计这 7 款软件从启动到界面稳定所需要的时间,并将该时间设置为超时时限.在相同的时间内,模拟模糊测试流程,检测两种方法运行程序的次数,以此来估算空转状态识别算法为模糊测试带来的效率,证明了本文提出的方法比传统时间阈值方法更加有效.

6.3 实验结果

6.3.1 验证实验结果

通过不同次数的学习对比,检测算法判断出程序进入空转状态时所经历的消息循环次数,并与程序本身的进入空转状态所经历的消息循环次数进行比较.通过使用不同大小的文件进行测试,检测工具能否有效地区分

待测程序的工作状态与空转状态;使用的执行迹条数表示语料库构造时用到的函数执行迹个数;TestAPP 记录结果表示程序完成测试用例处理后进入空转状态时所经历的消息循环的个数,该功能在 TestAPP 源码上实现,其记录结果表示真实情况;工具实验结果表示工具记录的程序完成测试用例处理后进入空转状态时所经历的消息循环的个数,每次测试完成后,会同时产出 TestAPP 记录结果与工具实验结果,保证测试环境相同.实验结果如图 15 所示.

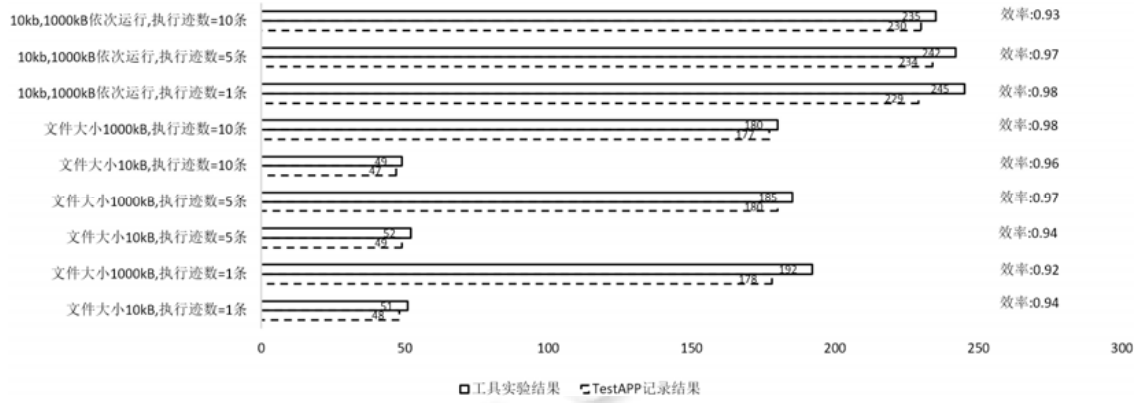


Fig.15 Graph of the result of the verification experiment

图 15 验证实验实验结果

由结果图 15 可知,使用的执行迹条数越多,生成的语料库就越能反映空转状态特征,算法测出的被测程序进入空转状态所经历的消息循环次数与实际所经历的消息循环次数也就越来越接近.同时,测试结果显示:识别出空转状态的时刻是进入空转状态后,表示工具的准确率高.不同大小文件的依次运行,表明虽然测试用例的复杂度不同,但工具仍能准确识别出程序进入空转状态的时机.随着构造语料库的函数执行迹的增多,工具的效率有一定程度的提高,说明构造丰富的语料库可以有效地提升工具的效率.

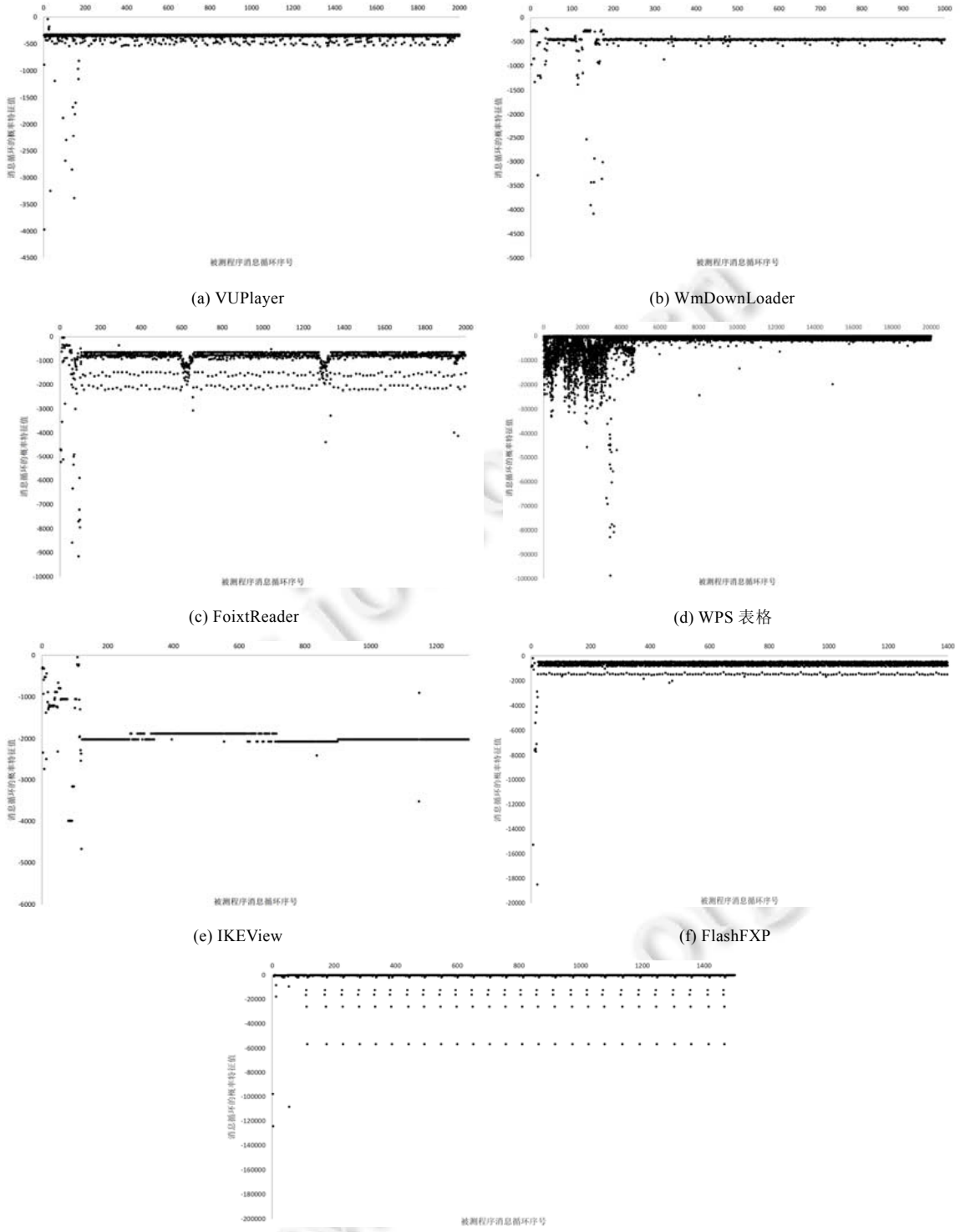
6.3.2 对比实验结果

分别针对 IKEView、WmDownLoader、VUPlayer、FoxitReader、WPS 表格、FlashFXP 以及 Dupsctc 这 7 种软件进行对比实验.IKEView 在读取文件前需要有 MessageBox 按钮点击确认过程;WmDownLoader 必须通过 GUI 交互完成文件输入;VUPlayer 可以使用命令行完成文件输入;FoxitReader 和 WPS 表格则是当前具有代表性的大中型商业软件;FlashFXP 以及 Dupsctc 则是从网路上获取数据并处理的.这 7 款软件中,IKEView, WmDownLoader,VUPlayer,FoxitReader 以及 FlashFXP 选用 *GetMessage* 函数作为分隔符;由于 WPS 表格以及 Dupsctc 采用 QT 编写,其消息循环依靠 *PeekMessage* 实现,选用 *PeekMessage* 作为分隔符.在进行测试前,每个程序加载测试用例,并等待 20 分钟记录函数执行迹,往复运行 10 次,得到语料库.

为了检测在测试过程中是否会出现误报问题,即待测程序进入空转状态前,进程是否被杀死,本文所选用的程序都有相应的 POC,通过运行 POC 检测崩溃界面是否出现来判断工具是否存在误报.

首先展示经过预处理后,每个程序从启动加载测试用例到关闭的消息循环的 $N(G)$值的散列图(如图 16 所示).由图 16 可知,这些程序的 $N(T)$图都具有如下特征:开始分布发散,而后聚敛.符合第 4 节的推断.

经过实际检测,被测程序在加载 POC 时,都正常弹出崩溃界面或者触发异常调试器,结果表明,工具具有高准确率.通过与超时时限方式进行对比,即执行相同多次数的程序,比较所消耗的时间.为了控制相应的变量,被测组采用 PinTool 实时检测工具进行加载,对比组采用 PIN 进行加载,通过观察的方式得到超时时限值.为了尽可能地还原工业界在超时时限方法的处理方式,即保证程序在完成测试用例用例处理后结束程序,本文超时时限的阈值时间的选择是随机测试 20 个大小不同的测试用例,选择处理时间最长的那个作为阈值时间.每个程序分别加载 300 个不同大小的测试用例,记录所需时间(单位:s),记录结果如图 17 所示.



(a) VUPlayer (b) WmDownLoader

(c) FoixtReader (d) WPS 表格

(e) IKEView (f) FlashFXP

(g) Dupsctc

Fig.16 Probability feature sequence of the programs from start to close

图 16 程序启动到关闭的消息循环概率特征序列

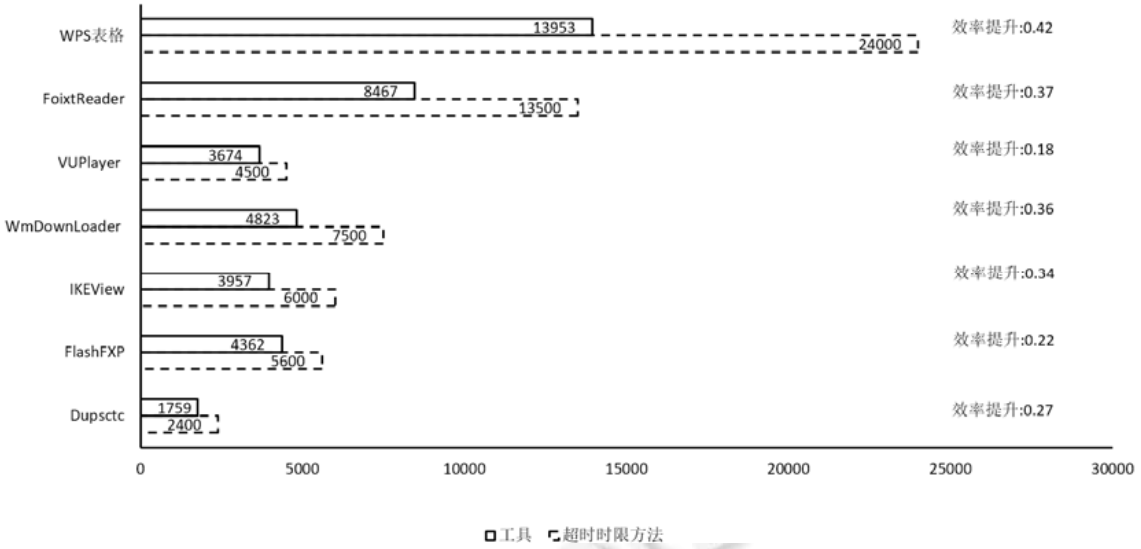


Fig.17 Graph of the result of the comparison experiment

图 17 对比实验实验结果

由图中数据可知,由于测试用例文件大小不一,因此使用超时时限方法时,为了避免文件加载完成前结束程序,需要选取耗时最长的时间作为超时时限参数时间,因此会出现测试用例已经加载完毕,但是仍需等待超时时限时间的情况,从而降低了测试效率.本文提供的工具则是通过对程序本身状态进行判断:若测试用例大,则执行时间长;若测试用例小,则执行时间短.因此在面对规模较大的程序时,能够准确地对程序进入空转状态的时机做出判断,从而提升测试的效率.

7 总 结

本文提出了基于函数执行迹的 Bi-Gram 模型来判断程序进入空转状态的时机,从实验结果看,原型工具在识别准确与效率上都比现阶段流行的超时时限方法优秀.接下来,根据原型工具,可以进一步应用到 Windows 下的 DBGENG 下,与现有的模糊测试工具中的后端异常检测相结合,进一步实用化,提升模糊测试效率.

同时,也可以将模型应用到除空转状态外的其他程序状态识别,例如测试用例文件加载或者测试用例文件处理等行为,这样可以减少人工程序分析的难度,从而提取程序关键功能区域.

References:

[1] http://www.freebuf.com/articles/neopoints/111712.html

[2] http://lcamtuf.coredump.cx/afl/

[3] Rawat S, Jain V, Kumar A, Cojocar L, Giuffrida C, Bos H. VUzzer: Application-Aware evolutionary fuzzing. In: Proc. of the Network and Distributed System Security Symp. (NDSS 2017). 2017. [doi: 10.14722/ndss.2017.23404]

[4] https://github.com/stephenfewer/grinder

[5] Tang ZG, Zhong MQ, Li HZ, Zhang J. File format vulnerability exploiting technique based on fuzzing. Computer Engineering, 2010,36(16):151−153. (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-3428.2010.16.055]

[6] Russinovich M, Solomon DA. Windows Internals: Including Windows Server 2008 and Windows Vista. Microsoft Press, 2009.

[7] Sharma A, Lyons J, Dehzangi A, Paliwal KK. A feature extraction technique using Bi-Gram probabilities of position specific scoring matrix for protein fold recognition. Journal of Theoretical Biology, 2013,320:41−46. [doi: 10.1016/j.jtbi.2012.12.008]

[8] Wu YL, Wei G, Li HZ. A word segmentation algorithm for Chinese language based on *N*-Gram models and machine learning. Journal of Electronics and Information Technology, 2001,23(11):1148−1153 (in Chinese with English abstract).

[9] Mao W, Xu WR, Guo J. A Chinese text classifier based on *N*-Gram language model and cha in augmented naive Bayesian classifier. Journal of Chinese Information Processing, 2006,20(3):29−35 (in Chinese with English abstract).

[10] Cavnar WB, Trenkle JM. *N*-Gram-Based Text Categorization. In: Proc. of the 3rd Annual Symp. on Document Analysis and Information Retrieval (SDAIR'94). Las Vegas, 1994. 161−175.

[11] Luk C, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K. Pin: Building customized program analysis tools with dynamic instrumentation. In: Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation. Chicago, 2005. 190−200. [doi: 10.1145/1065010.1065034]

附中文参考文献:

[5] 唐彰国,钟明全,李焕洲,张健.基于Fuzzing的文件格式漏洞挖掘技术.计算机工程,2010,36(16):151−153. [doi: 10.3969/j.issn.1000-3428.2010.16.055]

[8] 吴应良,韦岗,李海洲.一种基于 *N*-Gram 模型和机器学习的汉语分词算法.电子与信息学报,2001,23(11):1148−1153.

[9] 毛伟,徐蔚然,郭军.基于 *N*-Gram 语言模型和链状朴素贝叶斯分类器的中文文本分类系统.中文信息学报,2006,20(3):29−35.

张兴(1992－),男,内蒙古包头人,博士生,主要研究领域为二进制软件自动挖掘与利用.

雷青(1968－),女,博士,教授,博士生导师,主要研究领域为现代通信技术,网络空间安全.

冯超(1983－),男,博士,讲师,主要研究领域为二进制软件自动挖掘与利用.

唐朝京(1962－),男,博士,教授,博士生导师,主要研究领域为现代通信技术,网络空间安全.