基于代码进化的恶意代码沙箱规避检测技术研究

梁光辉 庞建民* 单 征
(解放军信息工程大学 郑州 450001)
(数学工程与先进计算国家重点实验室 郑州 450001)

摘 要：为了对抗恶意代码的沙箱规避行为，提高恶意代码的分析效率，该文提出基于代码进化的恶意代码沙箱规避检测技术。提取恶意代码的静态语义信息和动态运行时信息，利用沙箱规避行为在代码进化过程中所产生的动静态语义上的差异，设计了基于相似度差异的判定算法。在7个实际恶意家族中共检测出240个具有沙箱规避行为的恶意样本，相比于JOE分析系统，准确率提高了12.5%，同时将误报率降低到1%，其验证了该文方法的正确性和有效性。

关键词：恶意代码；沙箱规避；静态相似度
中图分类号：TP309 **文献标识码**：A **文章编号**：1009-5896(2019)02-0341-07
DOI: 10.11999/JEIT180257

Malware Sandbox Evasion Detection Based on Code Evolution

LIANG Guanghui PANG Jianmin SHAN Zheng
(*PLA Information Engineering University*, *Zhengzhou* 450001, *China*)
(*State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Zhengzhou* 450001, *China*)

Abstract: In order to resist the malware sandbox evasion behavior, improve the efficiency of malware analysis, a code-evolution-based sandbox evasion technique for detecting the malware behavior is proposed. The approach can effectively accomplish the detection and identification of malware by first extracting the static and dynamic features of malware software and then differentiating the variations of such features during code evolution using sandbox evasion techniques. With the proposed algorithm, 240 malware samples with sandbox-bypassing behaviors can be uncovered successfully from 7 malware families. Compared with the JOE analysis system, the proposed algorithm improves the accuracy by 12.5% and reduces the false positive to 1%, which validates the proposed correctness and effectiveness.

Key words: Malware; Sandbox evasion; Static similarity

1 引言

恶意代码数量的快速增长使得传统安全分析技术面临巨大的挑战，据2017年VirusTotal统计，全年新发现恶意代码1.4×10^7万种。为了应对海量恶意代码分析任务，同时有效对抗恶意代码中频繁使用的加壳、混淆等抗分析手段，自动化沙箱分析技术逐渐得到了广泛的应用。自动化沙箱技术通过虚拟化或全系统模拟技术建立一个真实可控的分析环境，让被分析代码在该环境中运行，并监控其运行过程，记录代码与系统之间的交互信息。自动化沙箱分析技术能够获取代码在运行过程中详细的行为信息，例如对文件、注册表、内存、进程和服务等操作系统资源的操作，可以让安全分析人员较为直观和快速地判断被分析的程序是否具有恶意性，具有较高的可靠性与易用性。在自动化沙箱分析技术的发展过程中，有多个典型的沙箱分析系统，例如Anubis, Temu, Cuckoo等。Anubis是第1个较为完善的在线沙箱分析系统，支持对可执行程序和URL的分析[1]。文献[2]开源了Bitblaze分析平台，其中的动态分析平台Temu提供了基于qemu的沙箱分析系统，先后有Renovo, Hookfind, Minesweeper等动态分析组件基于Temu沙箱开发并得到广泛应用[2]。Cuckoo沙箱作为谷歌公司的一个开源项目，同时支持对Windows, Linux和Android恶意代码的分析[3]。动态沙箱技术的不断发展，使安全分析人员能够从恶意代码实际的行为层面对其进行恶意性

收稿日期：2018-03-21；改回日期：2018-11-06；网络出版：2018-11-14
*通信作者： 庞建民 jianmin_pang@126.com
基金项目：国家自然科学基金(61472447, 61802435, 61802433)
Foundation Items: The National Natural Science Foundation of China (61472447, 61802435, 61802433)

判定，有效降低了误报率，而且动态沙箱能有效对抗恶意代码中常见的花指令、加壳、混淆等抗静态分析手段。

基于沙箱分析系统广泛部署所带来的影响，恶意代码制作者也研究了对应的抗沙箱分析手段，来逃避或绕过沙箱对自身的分析。以往的沙箱规避技术主要在一些复杂的恶意代码中才会出现，如某些定向攻击或者APT恶意代码，例如攻击工业控制系统的Shamoon使用了多种技术来规避Anubis沙箱的分析[4]。但是，随着沙箱规避技术的不断发展，沙箱规避技术在恶意代码中出现的比例越来越高，并逐渐成为了恶意代码的一种主流配置。据卡巴斯基统计，当前发现的恶意代码中，38%出现了不同程度的沙箱规避行为。勒索软件Locky从2016年出现以后，在其后续的变种中先后出现了反hook、反虚拟机和沙箱环境检查等行为，据Lastline分析报告，攻击金融类目标的carbanak, dyre, tinba 3种恶意代码家族也从2015年开始采用沙箱规避技术。沙箱通过虚拟化技术提供可控分析环境的同时，自身不可避免地会存在一些和实际物理机器不一致的特征，例如CPU执行效率、磁盘属性、进程特征等等。恶意代码制作者在发现这些不一致特征之后，在代码中加入了相应的对抗技术，从而降低自身被发现的几率。

Yokoyama等人[5]对20个提供恶意代码分析服务的系统平台进行了调查，通过分析生成的2666个分析报告，识别出了其中76个沙箱特征，而且发现只是基于沙箱的固有特性，就可以简单地检测和规避所有的沙箱。Kirat等人[6]设计了一种快速且无需重启的裸机系统，42个在虚拟或者模拟环境中没有行为的恶意代码在这个裸机系统中释放出了恶意行为，但是该裸机系统只能对部分系统状态进行还原，并且无法应对延迟执行等规避手段。Crandall等人[7]提出一种专注于检测具有特定规避行为的方法，例如只在某个特定时间被唤醒的时间逻辑炸弹。为了标记出恶意代码中所采用的具体的规避行为，Kirat等人[8]提出了在程序数据流的辅助下，利用生物信息学算法来定位系统调用序列中的规避行为，对2810个具有规避行为的恶意代码根据不同的规避行为进行了聚类，并且最终从中抽取了78种规避特征。Gilboy[9]提出了一种探索代码段的多路径执行方法，能够执行恶意程序中未被执行的代码片段，最终发现1000个恶意代码中有167个具有隐藏行为和沙箱规避行为。Tanbe[10]展示了一种基于web的侦察策略，使得在定向攻击中定制化的恶意代码能够规避沙箱而只在特定的目标系统上发作，作者认为类似的方法将来有可能会被更多的恶意代码使用。

Kruegel[11]在2015年RSA大会上提出，搭建一个沙箱系统很容易，但是搭建一个有效的沙箱系统很难，因为对抗沙箱的技术实现过程较为隐蔽，很难被检测和识别，而且这种技术处于不断的变化和更新之中，之前的检测规则随着恶意代码的进化可能就会失效。

基于此，本文提出从代码进化角度来检测沙箱规避行为的方法，本文的贡献如下：

(1) 提出了一种基于代码进化角度的恶意代码沙箱规避技术检测模型，可以实现在单沙箱配置情况下对具有沙箱规避的恶意代码的检测。

(2) 基于静态和动态分析方法，分别提取二进制程序的静态子程序特征和动态系统调用序列特征，提出了基于动静态语义差异的规避行为检测算法。

(3) 在实际的恶意代码家族上实现了沙箱规避检测方法，验证了本方法的正确性和有效性。

本文内容安排如下：第1节是引言，介绍了研究背景及国内外研究进展；第2节是沙箱规避技术，对沙箱规避技术进行分类介绍；第3节是系统实现，详细介绍了从进化角度来判定恶意代码中的沙箱规避行为；第4节是实验评估，通过对实际捕获的恶意代码家族样本的分析，来评估本技术的实际效果；第5节对全文进行总结。

2 沙箱规避技术

沙箱规避技术主要是指恶意代码为了逃避动态沙箱系统的分析，在释放真正的恶意行为之前，首先通过读取系统的某些特征，检查当前所处的环境是否位于可能被分析的沙箱环境之中，如果位于沙箱中，恶意代码就会自动退出或执行一些无害的行为，如果没有检测到沙箱分析环境，才会释放真实的攻击载荷。

2.1 用户交互特征

自动化的沙箱分析系统用来进行大规模恶意代码的批处理分析，从恶意代码的载入，运行到后来行为的监控，报告的生成都是自动化地实现，整个过程很少需要用户的介入。基于此，一些恶意代码通过判断当前用户与系统之间的交互来判断是否位于沙箱中。

木马BanerChan在检测到一定数量的鼠标点击时再启动自身的载荷，然后才与C&C服务器建立通讯连接；某些嵌入在office文档中的恶意代码，会检测当前文档是否滚动到第2页，在典型沙箱中，模拟的鼠标滚动都是通过随机产生的，很难产生类似人类行为的鼠标滚动行为。另外，恶意代码

还可以通过计算鼠标移动的速度来判断鼠标是否由用户实际控制。

2.2 主机特征

沙箱分析系统需要通过虚拟化技术搭建一个虚拟的操作系统，这个虚拟的环境所具有的特征成为很多恶意代码判断沙箱的重要条件。恶意代码通过检测当前主机的特征来判断是否处于虚拟环境中，例如主机的用户名、安装的软件列表、系统文件、CPU个数、BIOS特征、MAC地址等，如果没有专门去清除这些特征，虚拟主机与真实主机在这些地方有着较为明显的区别。例如恶意软件通过查询注册表项“HKEY_LOCAL_MACHINE\HARDWARE\DescriptionSystem\”以查找与常见虚拟机运行(如VMWare)相关联的值。在redpill工具中，通过VPCEXT指令能够检测VirtualPC的痕迹，文献[12]提出通过收集系统的使用时间和人为磨损程度等信息，建立了一个决策树分析模型来检测沙箱环境，达到了92%的检测精度。

除了沙箱的虚拟特征之外，某些行为监控工具的痕迹也会被恶意代码利用，例如恶意代码会通过进程遍历来查找某些敏感的进程，如监控进程、内核hook进程，也可以通过窗口遍历来发现是否有文件监控、注册表监控对应的软件界面窗口。

2.3 时间开销

虚拟化主机除了存在某些可检测的主机特征外，在时间开销上与真实的主机也存在差别。在仿真或者虚拟的过程中，虚拟主机不可能分配到物理系统的全部资源，而且为了实现动态行为监控，需要加载一些插桩模块或监控驱动，这样使得虚拟主机CPU的实际处理能力要低于真实物理主机。在redpill工具中，利用特权指令RDTSC返回CPU的时间戳计数器，通过两次RDTSC指令之间的时间周期来判断是否符合物理主机的预期值。

除了利用虚拟主机与真实主机的性能差异，时间开销规避的另一种就是休眠。沙箱为了能够处理大量的样本，在分析每个恶意代码时都设置了固定的分析时间，如果超过所设置的分析时间，被分析样本还没有结束，则沙箱会强制退出，恢复快照，继续分析下一个样本，基于此，恶意代码会故意在某些关键的行为之前设置一段时间的休眠时间，从而逃避沙箱的监测。

除了上面3类规避技术之外，恶意代码还可以使用命令行参数等技术来规避沙箱分析。沙箱规避技术的一个重要特点就是变化速度非常快，而且一种恶意代码会同时使用多种技术相互配合来达到规避分析的目的。所以，要对抗这些具有规避功能的恶意代码，一方面需要不断地改进沙箱的功能，使其更接近于真实的物理主机，另一方面，沙箱规避技术的检测也是一个非常重要的研究方向。

3 系统实现

本节首先介绍了如何从恶意代码的进化视角看分析沙箱规避技术，然后基于静态分析方法，分别提取恶意代码中子程序的操作码特征和反编译特征，提出相应的相似度计算方法，借助层次聚类实现了对恶意代码家族的聚类，最后再通过动态特征的抽取，表征恶意代码在动态沙箱下的行为，并利用沙箱规避恶意代码在动态和静态环境中的语义差异，设计了响应的规避行为检测算法。

3.1 代码进化视角的规避技术

规避技术并不是恶意代码的天生特性，恶意代码在和安全分析技术的对抗中会不断进行自我进化和革新，添加新的功能片段，舍弃旧的模块，一方面是为了躲避安全产品的分析检测，另一方面为了不断完善自身来适应新的应用场景。沙箱规避技术的出现也是这样，具有沙箱检测的功能片段被添加到原有的恶意代码之中，新的代码具有了沙箱规避的能力。代码的进化的过程如表1所示。

表 1　沙箱规避代码进化示意

Malicious code A	Evasive Malicious code B
Main_behavior( )	Main_behavior( )
{	{
1 Registry_opeartion( )	1 flag = check_sandbox( )
2 Process_injection( )	2 if (flag == True):
3 File_compress( )	3 do_benign( ) or exit()
4 Connection_C&C_server( )	4 else:
5 Waiting_for_instruction( )	5 Registry_opeartion( )
6 File_send( )	6 Process_injection( )
}	7 File_compress( )
	8 Connection_C&C_server( )
	9 Waiting_for_ instruction( )
	10 File_send( )
	}

恶意代码A通过注册表操作、进程注入、文件压缩、网络连接等一系列行为实现自身的恶意行为，恶意代码B在不改变恶意行为的前提下，加入了沙箱检测代码，如果沙箱检测为真，则进行良性操作或者结束自身，如果沙箱检测为假，才释放真正的恶意行为。可以发现，恶意代码通过加入规避代码实现了对沙箱的检测，但是其核心的恶意功能并没有改变，规避代码相对于整个恶意行为的实现

代码来说，代码量占比重较小，也就是从代码静态相似度来说，恶意代码A和恶意代码B具有相当高的相似性。

在代码的进化过程中，除了从无沙箱规避行为向有沙箱规避行为的进化，还包括从初级规避行为向复杂规避行为的进化，这两种进化过程，都是规避行为不断丰富的过程。

基于此，本文提出了基于代码进化的恶意代码沙箱规避检测方法。从恶意代码进化的角度出发，分别提取恶意代码的静态特征和动态特征，通过分析其在不同特征下的相似性，检测和识别恶意代码中的沙箱规避行为。该方法对应的模型框架如图1所示，恶意代码首先通过多个杀软标签实现家族的分类，然后同一家族的恶意代码分别进行静态语义的分析和动态行为的分析，并计算对应的相似度，最终利用动静相似度之间的差异，实现沙箱规避行为的判定。

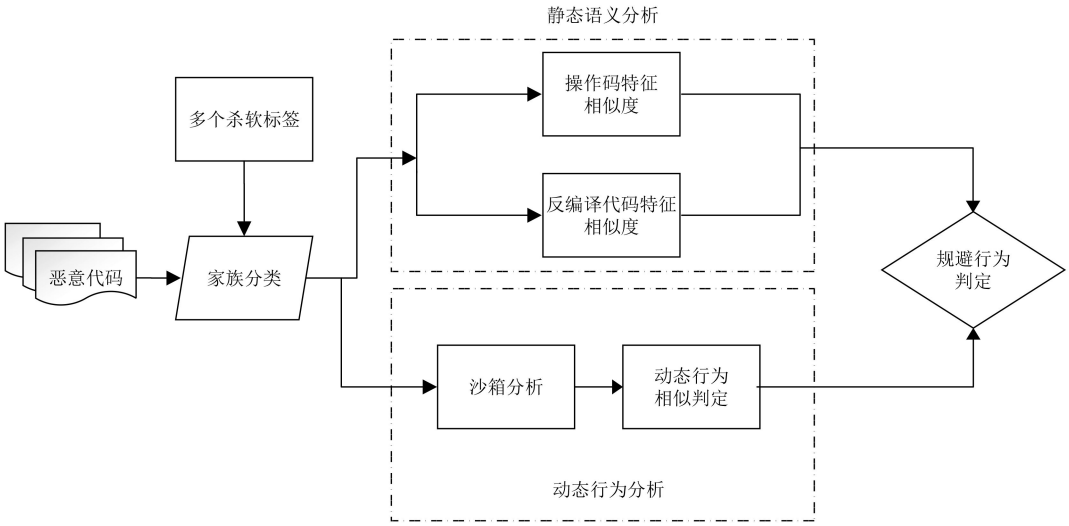

图 1 模型框架

3.2 基于静态语义的相似度

基于子程序(Subroutine)的特征比对逐渐被应用于二进制程序漏洞挖掘与恶意代码分析。Bindiff是谷歌公司开源的一个商业化的分析产品，能够比对两个基于相同编译器生成的二进制程序，但是其一次只能分析一对二进制程序，无法应用于大规模的恶意代码分析[13]。Steven使用二进制程序中的字符串和Subroutine分别实现了对样本的聚类和子程序的聚类。Joseph提出了基于子程序分析的APT检测方法，他认为以往对恶意代码的检测，忽略了未知代码与已知恶意代码在某些子程序上的相似，从而造成了漏报，通过分析子程序的相似性，能够提高对APT恶意代码的检测精度。

3.2.1 子程序特征提取

首先对二进制程序的代码段行进递归的反汇编扫描，然后在汇编程序中识别不同子程序的边界。一个子程序是一个功能相对较为完整的指令执行集合。从子程序的两种形式分别提取程序的静态语义信息，这两种形式分别是操作码序列(Opcode)I，反编译代码D。

操作码指示了计算机所要执行具体操作的机器指令，包括算数运算、数据操作、逻辑操作和程序控制等，是程序语义在底层的具体体现，本文通过2-gram算法来生成每个子程序的操作码特征，对于第k个子程序，其2-gram操作码特征由I_k来表示；

反编译是指将可执行程序从一个相对较低的抽象层次(机器可理解)转化为一个较高的抽象层次(人可理解)的过程，例如静态分析工具IDA提供的反编译插件HexRays，实现了从汇编指令到类C程序的转化，使得分析人员能够从一个较高层次的语义上来理解程序行为。张一弛[14]基于恶意代码的反编译代码，实现了对未知恶意代码的检测。本文借助于HexRays插件来生成子程序的反编译代码，对第k个子程序，生成后的反编译代码用D_k来表示。

基于此，二进制程序中的第k个子程序S_k的特征表示为

$$S_k = \langle I_k, D_k \rangle \tag{1}$$

两个子程序S_k和S_t之间的相似度可以用如式(2)的情况表示

$$P(S_k, S_t) = \frac{P(I_k, I_t) + P(D_k, D_t)}{2} \tag{2}$$

其中，$P(I_k, I_t)$表示操作码特征的相似度，$P(D_k, D_t)$表示反编译代码的相似度，根据这两种特征的特

点，分别应用了不同的相似度计算算法，对于操作码特征，采用Jaccard相似度

$$P(I_k, I_t) = \frac{|I_k \cap I_t|}{|I_k \cup I_t|} \tag{3}$$

对于反编译代码相似特征，采用了模糊哈希算法来计算两个反编译代码片段的相似度

$$P(D_k, D_t) = \text{Fuzzy\_hash}\,(D_k, D_t) \tag{4}$$

3.2.2 静态相似度计算

对于恶意代码$A = \{S_{A1}, S_{A2}, \cdots, S_{AK}\}$和恶意代码$B = \{S_{B1}, S_{B2}, \cdots, S_{BK}\}$，分别比较$A$中子程序与$B$中子程序的相似性。为了提高比较的效率，先将程序中的子程序进行排序，排序的依据是子程序中所包含的基本块的数量。

则，恶意代码A和B之间的静态语义相似度PS(A, B)计算如式(5)

$$\text{PS}\,(A, B) = \frac{\sum_{j=1}^{t}\sum_{i=1}^{k}\max \text{PS}\,(S_{Ai}, S_{Bj})}{t + k} \tag{5}$$

3.3 基于动静结合的规避行为检测

3.3.1 动态行为监控

为了记录恶意代码在沙箱中的实际运行轨迹，本文选择了Cuckoo沙箱。Cuckoo沙箱2012年由谷歌“Summer Code”计划推出，经过多年的维护和不断升级，已经成为当前主流的开源沙箱。为了更好模拟真实主机环境，在Guest操作系统上安装了office文档处理、Adobe阅读器、影音播放软件等常见的应用软件。

Cuckoo通过注入到被分析的恶意代码的进程空间，实现了对恶意代码动态运行过程的追踪，并定义了200多个相关的系统调用，涉及到文件操作、注册表操作、网络通联、服务操作和系统交互等相关的行为。监控过程以进程为基本单位，恶意代码在实际感染系统过程中，会创建多个进程来完成不同的功能，最终会形成以进程为单位的行为轨迹。本文根据进程创建的先后顺序，将多个进程行为轨迹合并为1条行为轨迹，以此作为恶意代码的最终行为轨迹。

恶意代码为了对抗某些监控工具，往往不调用WIN32 API，而是直接调用底层的系统，Cuckoo在行为监控的时候也通过SSDT对这些底层的系统调用进行了监控，这种监控模式就导致了最终形成的行为轨迹会非常的冗长，KI等人[15]在基于Dynamo RIO的监控中也发现了这种现象。基于此，本文对合并轨迹中重复的系统调用进行了去重，精简了行为轨迹。

3.3.2 动态特征抽取与相似性

本文采取3-gram对恶意代码的动态行为轨迹进行抽取，每个样本的动态行为特征表示为

$$B = (gs_1, gs_2, \cdots, gs_n) \tag{6}$$

其中，gs_i表示第i个3-gram片段，gs_1的取值为0或1，0表示不存在这个行为序列片段，1表示存在这个行为序列片段。

特征之间的相似度计算由Jaccard相似度表示

$$P(B_k, B_t) = \frac{|B_k \cap B_t|}{|B_k \cup B_t|} \tag{7}$$

相似度为1时，表示两个样本的行为完全一致，相似度为0时，表示两个样本的行为完全不一样。

3.3.3 规避行为检测算法

检测的样本集合记为$M = \{M_1, M_2, \cdots, M_N\}$，两个样本之间的静态语义相似度记为PS，两个样本之间的动态行为相似度记为PB，对比行为检测算法如表2。

表 2　规避行为检测算法

For (M_i, M_j) in C_M^2:
  PS $(M_i, M_j) = \alpha$
  if $\alpha < \varepsilon$:
    continue
  PB $(M_i, M_j) = \beta$
  if $\alpha - \beta > \tau$:
    if $B_i > B_j$:
      M_j is an evasive malware
    else:
      M_i is an evasive malware
  else:
    No Evasion

检测算法中，ε为恶意代码之间静态相似度的阈值，τ称为恶意代码的静动态语义的差异阈值，根据大量手工分析，本文将阈值ε的初始值设为0.8，阈值τ的初始值设置为0.6。

4　实验评估

4.1 实验准备

本文从MalwareBenchmark恶意代码库中选取了7个恶意代码家族[16]。分别为Bifrose, Dridex, Necurs, Sfone, Unruy, Urleas, Confidence家族，为了降低加壳对静态语义信息带来的影响，对家族中加壳程序进行过滤后，共计得到1627个恶意代码。这7个家族的恶意代码分布时间从2008年—2016年。各个家族的恶意代码数量统计如表3所示。

表 3 恶意代码家族情况

家族名称	数量	时间分布
Bifrose	317	2008～2015
Dridex	57	2012～2014
Necurs	154	2011～2016
Sfone	151	2009～2016
Unruy	725	2008～2016
Urleas	57	2010～2015
Confidence	166	2011～2016

实验平台中Host系统为Ubuntu 16.04操作系统，32 G内存，1 T硬盘，动态沙箱Guest系统为Windows7 32位操作系统。

4.2 实验结果与评估

将7个恶意代码家族总计1627个恶意代码加载到本文提出思路实现的系统MSED(恶意代码沙箱规避检测系统)中，发现了240个具有沙箱规避行为的恶意代码，各家族的统计情况如表4所示。

表 4 沙箱规避情况统计

家族名称	样本数量	沙箱规避样本数量	百分比(%)
Bifrose	317	41	12.9
Dridex	57	22	38.5
Necurs	154	40	25.9
Sfone	151	11	7.2
Unruy	725	70	9.6
Urleas	57	30	52.6
Confidence	166	26	15.6

表4中，沙箱规避行为占比最高的为Urleas家族，最低的为Sfone家族。通过对这240个由MSED沙箱规避样本进一步的手工分析，发现其中226个样本都具有沙箱规避行为，并且大部分样本同时具有两种以上的规避行为。本文将发现的规避行为大致分为7种，分别为虚拟主机特征检测、进程查找、交互行为检测、睡眠对抗、时间开销检测、沙箱环境检测、特权指令。

通过图2可以发现，恶意代码使用最多的3个规避技术是虚拟主机特征检测、睡眠对抗和进程服务查找。在226个样本中出现了510次的虚拟主机特征检测，人工分析发现，这是因为同一个恶意代码会通过多次不同的行为来检测虚拟环境，例如会先后通过注册表键值查询、配置文件查找、互斥量检测等来判断是否处于虚拟环境。在睡眠对抗分析中，大部分样本的睡眠时间会超过300 s，这个时间设置也超过了大部分沙箱分析的时间。在进程服务查找中，恶意代码会检测当前的进程列表或者服务列表中是否会有VboxService, Vmware, VirtualPC等虚拟进程或服务。

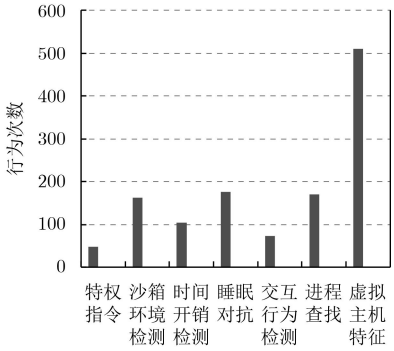

图 2 沙箱规避行为统计

为了检测对沙箱规避行为检测的准确性，本文将分析结果与JOE动态分析结果做了对比。JOE分析系统是一个公开的商业分析系统，其中设置了大量的沙箱规避特征和启发式规则，来检测恶意代码中的沙箱规避行为，并将检测到的规避行为标记打分，最终通过综合评分来判断恶意代码是否具有沙箱规避行为[17]。

本文的MSED系统检测到240个样本，人工分析后发现其中226个样本具有沙箱规避行为，其中14个样本未发现规避行为，经过进一步分析发现这14个样本均存在2种私有壳。这两种私有壳影响了程序静态语义信息的正确性，造成了误报。

将240个样本加载到JOE系统中，其中191个被标记为Evader，即沙箱规避样本。另外将MSED标记为不具有的沙箱规避行为的1387个样本加载到JOE系统中，93个被标记为Evader的样本有52个在JOE系统中表现出了明显的恶意行为，有37个在MSED系统中表现了明显的恶意行为。MSED与JOE的检测对比结果如表5所示。

通过上述分析发现，本文提出的MSED系统，其精确率和FPR要高于JOE分析系统，查全率TPR要低于JOE分析系统，主要是因为受所采集样本范围的限制，没有收集到相应的进化前或进化后对比样本，如果进一步扩大样本采集的范围，将能够在一定程度上提高TPR。

5 结束语

本文提出了基于代码进化的恶意代码沙箱规避

表 5 检测结果对比

	TP	TN	FP	FN	精度(%)	TPR (%)	FPR (%)	F1
JOE	232	1294	52	49	81.6	82.5	3.8	0.8204
MSED	226	1331	14	56	94.1	80.1	1	0.8653

检测技术，利用恶意代码在进化过程中引入沙箱规避的特点，获取恶意代码的动静态语义信息，计算同一家族内不同恶意代码之间的动静态语义差异，提出了对应的检测算法，并在实际的7个恶意代码家族中进行了实验，实验表明，本文提出的MSED检测方法，相比于现在的商业应用系统，能够更有效地发现恶意代码中的沙箱规避行为，同时能够保持较低的误报率。

下一步，一方面可以将目前已经发现具有规避行为的恶意代码作为重点对比对象，随着沙箱的升级和更高级的沙箱对抗行为的出现，就可以将之前已经判明的恶意样本作为种子来挖掘新的沙箱规避样本，这样就可以形成对沙箱规避恶意代码的持续检测。另一方面，研究通用脱壳技术，能够对部分私有壳进行分析与脱壳处理，提高本方法的处理效率。

参 考 文 献

[1] ANUBIS: Analyzing unknown binaries[OL]. www.anubis.iseclab.org, 2015.

[2] YIN Heng and SONG Dawn. Temu: Binary code analysis via whole-system layered annotative execution[R]. Submitted to Vee University of California, Berkeley, Tech Rep, 2010.

[3] CUCKOO Sandbox. Automated malware analysis[OL]. www.cuckoosandbox.org, 2016.

[4] RAIU C, HASBINI M, BEOLV S, *et al*. From Shamoon to Stonedrill-Wipers attacking Saudi organizations and beyond[R]. Kaspersky Lab, March, 2017.

[5] YOKOYAMA A, ISHII K, and TANABE R. SandPrint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion[C]. International Symposium on Research in Attacks, Intrusions, and Defenses, SudParis, France, 2016: 165–187.

[6] KIRAT D, VINGA G, and KRUEGEL C. Barebox: Efficient malware analysis on bare-metal[C]. Proceeding of the 27th Annual Computer Security Applications Conference, Orlando, USA, 2011: 403–412.

[7] CRANDALL J R, WASSERMANN G, and OLIVEIRA D A S. Temporal search: Detecting hidden malware timebombs with virtual machines[J]. *ACM SIGARCH Computer Architecture News*, 2006, 34(5): 25–36. doi: 10.1145/1168919.

[8] KIRAT D and VIGNA G. MalGene: Automatic extraction of malware analysis evasion signature[C]. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, USA, 2015: 769–780.

[9] GILBOY M R. Fighting evasive malware with DVasion[D]. [Master dissertation], University of Maryland, College Park, 2016: 31–44.

[10] TANBE R. Evasive malware via identifier implanting[C]. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Pairs, France, 2018: 162–184.

[11] KRUEGEL C. Evasive malware exposed and deconstructed[C]. RSA Conference, San Francisco, USA, 2015: 112–120.

[12] MIRAMIRKHANI N, APPINI M P, and NIKIFORAKIS N. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts[C]. IEEE Symposium on Security and Privacy, San Jose, USA, 2017: 1009–1024.

[13] BINDIFF[OL]. www.zynamics.com/bindiff.html. 2017.

[14] 张一弛. 基于反编译的恶意代码检测关键技术研究与实现[D]. [博士论文], 解放军信息工程大学, 2009: 22–39.
ZHANG Yichi. Research and Implementation of critical technology in malware detection based on decompilation[D]. [Ph.D. dissertation], PLA Information and Engineering University, 2009: 22–39.

[15] KI Y, KIM E, and KIM H. A novel approach to detect malware based on API call sequence analysis[J]. *International Journal of Distributed Sensor Networks*, 2015, 58(7): 3201–3206.

[16] MALWAREBENCHMARK[OL]. www.malwarebenchmark.org, 2018.

[17] JOESECURITY Sandbox[OL]. www.joesandbox.com, 2018.

梁光辉：男，1987年生，博士生，研究方向为恶意代码分析.

庞建民：男，1964年生，教授，博士生导师，研究方向为网络安全、先进计算.

单　征：男，1977年生，教授，博士生导师，研究方向为网络安全.