一种抗混淆的恶意代码变种识别系统

王 蕊[1,2],苏璞睿[2],杨 轶[2,3],冯登国[1,2]
(1.中国科学院研究生院信息安全国家重点实验室,北京 100049;
2.中国科学院软件研究所信息安全国家重点实验室,北京 100190;
3.信息安全共性技术国家工程研究中心,北京 100190)

摘 要: 恶意代码变种是当前恶意代码防范的重点和难点.混淆技术是恶意代码产生变种的主要技术,恶意代码通过混淆技术改变代码特征,在短时间内产生大量变种,躲避现有基于代码特征的恶意代码防范方法,对信息系统造成巨大威胁.本文提出一种抗混淆的恶意代码变种识别方法,采用可回溯的动态污点分析方法,配合触发条件处理引擎,对恶意代码及其变种进行细粒度地分析,挖掘其内在行为逻辑,形成可用于识别一类恶意代码的特征,并通过特征融合优化以及权值匹配等方式,提高了对恶意代码变种的识别能力.通过实验,验证了本文的识别方法对恶意代码及其混淆变种的识别能力.

关键词: 恶意代码变种;动态污点分析;行为分析;混淆技术
中图分类号: TP302.7 **文献标识码:** A **文章编号:** 0372-2112 (2011) 10-2322-09

An Anti-obfuscation Malware Variants Identification System

WANG Rui[1,2],SU Pu-rui[2],YANG Yi[2,3],FENG Deng-guo[1,2]
(1. *State Key Laboratory of Information Security*, *Graduate University of Chinese Academy of Sciences*, *Beijing* 100049, *China*;
2. *State Key Laboratory of Information Security*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*;
3. *National Engineering Research Center for Information Security*, *Beijing* 100190, *China*)

Abstract: Malware variants are one of the major challenges in malware detecting today. Obfuscation, as a most popular technology to generate these variants, can change the signatures of malware to avoid the current signature-based malware preventing method, which is a big threat to information system. This paper proposes a novel anti-obfuscate malware detecting method. By making use of dynamic taint analysis methods and trigger-based behavior processing engine, this method can abstract the essential behavior logic of malware in fine-grained and form it as signatures of a class of malware, and identify variants more precisely associated with signature merging optimizing process and fuzzy matching methods. Experiment results show that the detecting method in this paper can identify malwares and its variants efficiently.

Key words: malware variants; dynamic taint analysis; behavior analysis; obfuscation

1 引言

随着信息技术的发展和信息系统应用的普及,其安全形势也日趋严峻.恶意代码是影响信息系统安全的主要因素之一,其防范已成为保障信息系统安全的重点,其中对恶意代码变种的识别是防范研究中的关键.

混淆技术(obfuscation)是恶意代码产生变种的主要技术.最简单的混淆技术是代码加密,通过变换核心代码的表现形式躲避检测.常用混淆技术还包括多态(Polymorphism)和变形(Metamorphism)技术等.多态技术对代码整体进行变换,隐藏其特征,但其反变换后的代码主体不变,利用虚拟或模拟技术可提取出不变的代码主体.变形技术则在保留主要功能的前提下,采用代码变换、垃圾插入等方式对代码主体进行变换.恶意代码技术不断发展,可以预见,恶意代码防护将面临诸多挑战.

基于特征的识别方法是目前恶意代码检测的主流方法.传统的特征技术从代码中提取一段序列作为特征,采用序列匹配的方式对目标代码进行识别.该方法识别过程简单、误报率低,但局限于原恶意代码样本,简单的混淆技术即可绕过.此后研究者提出基于行为特征的分析方法,识别恶意代码的实际行为而非代码结构标记,从而抵抗加密等代码层混淆技术的干扰.然而,如何表征恶意代码关键行为、提高对变种的识别能力仍是目

收稿日期:2010-09-08;修回日期:2010-12-20
基金项目:国家863高技术研究发展计划(No.2009AA01Z435);国家自然科学基金(No.60703076,No.61073179)

前有待进一步研究的问题.

本文提出一套抗混淆的恶意代码变种识别系统，该系统包括行为分析、特征提取和恶意代码识别三个主要部分.本方法基于硬件模拟平台对恶意代码的执行流程实现高透明度监控，通过污点分析技术细粒度分析代码行为，提取关键行为逻辑，并针对常用混淆技术进行抗干扰处理，构建行为特征，以此为基础，监控目标代码行为，与特征进行匹配，通过特征融合优化、权值判定等方法，实现对恶意代码变种的识别.

本文的主要贡献如下：

(1)提出了一种恶意代码特征提取及描述方法，改进了传统的行为特征，并针对常用混淆方法对特征进行了抗混淆处理，提高了特征描述能力以及对变种的匹配能力；

(2)提出了一种基于行为特征的恶意代码变种识别方法.提出了特征融合等优化算法，基于融合的特征对恶意代码及其变种进行识别，改善了匹配效率并缩减了特征存储数据量；

(3)实现了一套抗混淆的恶意代码变种识别原型系统.在系统中设计了恶意代码常用触发条件处理引擎，提高了动态监控分析的全面性和准确性.通过实验评估，验证了本方法对恶意代码及其变种的识别能力.

2　相关工作

动态污点分析(dynamic taint analysis)[2]是近年来提出的一种有效跟踪及分析程序数据处理流程的分析方法，具有分析粒度细、准确性高及不依赖于源代码等特点，在恶意代码防范领域取得了一系列的研究成果.如 Egele 等人[5]使用动态污点技术分析检测间谍软件行为.Panorama 系统[3]采用硬件层污点技术对恶意代码进行分析，通过引入敏感信息并标记为污点，在硬件层监控污点传播过程并记录代码对敏感数据的访问和处理过程，生成污点图，然后基于污点图定义各种策略识别恶意代码.

恶意代码检测可分为特征检测(signature-based detection)和启发式(heuristics-based detection)检测两种.启发式检测可识别新恶意代码样本，但其通常基于一些启发元素而非恶意代码的本质特性，可能引发高误报率.特征检测方法[8~10]分析已知恶意代码样本并提取相关特征，通过匹配特征来识别恶意代码，具有误报率低、效率高等特点.传统的特征检测基于代码特征，易受混淆技术干扰，对恶意代码变种只能通过新建特征进行识别，随着变种大量出现，特征库数据量对存储和效率构成潜在威胁.基于混淆技术改变代码结构形式、隐藏代码特征但不改变行为的特点，研究者从代码执行的行为着手[4,5,9]，提取行为特征，降低混淆技术的干扰.行为特征的表示方法多样，如 Bonfante 等[9]使用控制流程图(CFG)表示行为特征，通过分析同构 CFG 对抗部分混淆；Christodorescu 等[4]使用 API 调用及数据依赖关系描述特征.为增强特征对变种的适应能力，研究者又提出特征的语义解析，如 Sathyanarayan 等[10]分析恶意代码中的 API 调用，结合语义，通过统计学方法进行检测，降低了部分代码混淆技术干扰，但其分析复杂度较高.

恶意代码为躲避检测，常在执行中加入一些触发条件.目前触发行为分析多依赖于人工，如 Crandall 等[15]提出一种基于虚拟机的恶意代码触发时间分析方法，利用虚拟机干扰时间的方式识别触发行为.Brumley 等[14]提出了支持多种触发类型的触发行为自动识别方法，通过具体与符号执行相结合的方法，分析路径分支条件挖掘触发行为并获取触发条件.该方法需用户确定触发类型以及相关输入在内存中的位置，在无先验知识的情况下，只能由系统辅助猜测完成.

3　问题描述及方法概述

当前恶意代码检测方法的主要问题在于对恶意代码变种的识别能力不足.恶意代码变种主要来自于混淆技术.

经分析，恶意代码的混淆技术通常包括垃圾代码插入、寄存器重分配、代码变换以及等价代码替换等.垃圾代码插入包括两种：一是在代码中插入一些不执行的无关代码干扰静态分析；二是插入一些会执行、但与恶意行为无关的代码.寄存器重分配是指为扰乱代码分析而将常用的寄存器用途改变.代码变换包括代码压缩、加密、加入跳转指令变换代码顺序、改变相互无关代码的执行顺序等.等价代码替换是将行为结果一致的不同代码互相替换，从而使利用指令或系统调用描述的行为特征产生变化.总的来说，混淆技术通过改变代码的语法表征，隐藏其内部逻辑关系，从而产生恶意代码变种，使当前的检测系统无法识别.

恶意代码还常通过加入触发条件实现自身的隐蔽而躲避分析.触发条件是指为恶意行为构建的一些执行条件，只有当条件满足时才执行相应行为.触发条件的主要功能包括：(1)降低恶意行为频率，从而降低恶意代码被识别的概率，如果在分析过程中没有满足触发条件，则无法发现恶意行为；(2)延后恶意行为的执行时间，使分析超时，从而躲避检测；(3)通过检查一些条件辨别代码是否处于分析环境之中，从而采取相应的反制手段.

基于以上分析，本文的思路是：动态监控恶意代码行为进行分析，提取关键行为特征并进行抗混淆处理，在此基础上，对恶意代码及其变种进行识别，通过特征

库优化、触发条件处理以及权值判定等方式，提高对恶意代码变种的识别能力和识别效率.

本文的恶意代码变种识别过程包括行为分析、特征构建和变种识别三个主要部分.

第一部分是行为分析.为了对恶意代码及其变种进行识别，首先要对恶意代码样本进行分析.本文采用污点分析技术分析恶意代码行为，并构建回溯流程挖掘行为关联性.采用动态分析恶意代码执行流程的方式，可避免混淆技术中阻碍反汇编、改变代码静态结构等方法的干扰，如插入不执行的垃圾代码、压缩加密等方式便不再影响分析效果.

第二部分是行为特征构建.对基于特征的识别方法，特征提取是决定识别能力和识别效率的重要因素.本文使用恶意代码的关键行为和行为之间的依赖关系构建行为特征，并以依赖图的形式进行描述.从系统的角度，可通过观察恶意代码对系统资源的使用情况识别代码行为[3].在方法实现上，行为监控与分析主要通过截获代码对系统 API 调用的使用实现.在系统中有大量 API 调用与恶意代码功能实现不相关，综合考虑效率和准确性，本文只关注与安全相关的 API 调用，即关键 API 调用(critical API calls)[10].由于在恶意代码使用各种混淆技术改变代码特征时，其数据演算和使用过程在不同变种之间保持相对稳定，因此使用数据依赖关系可以很好地表现恶意代码的行为关联.本文通过数据流和控制流分析，提取数据依赖关系，并恢复被控制依赖切断的数据依赖.在特征描述方面，使用关键 API 调用作为节点，完整的数据依赖关系作为边，以依赖图描述特征，与传统序列描述方法相比，对恶意代码行为具有更强的描述能力.

第三部分是恶意代码及其变种识别.在系统初始化时，读入恶意代码行为特征库，对其中的特征图进行优化，然后动态监控目标代码，与特征进行匹配.本文使用基于硬件模拟器的平台，在不影响真实系统的同时实现对代码的高透明度监控，防止被恶意代码察觉而采取反制手段.在监控过程中，通过动态污点分析技术对代码执行单步指令分析以及 API 调用拦截，识别关键行为数据，若实时数据与相关特征达到一定的匹配程度，则表明识别出了恶意代码或其变种.由于恶意代码种类日益增多，如果匹配针对单个特征进行，则遍历特征库中的大量特征需要很高的时间及空间复杂度.为了简化匹配算法、提高效率，本文对特征进行融合优化处理.由于混淆技术的影响，不同变种会出现 API 调用或依赖关系的变化，严格匹配的判定方法会导致漏报，本文采用权值判定的方法，记录匹配数据，计算匹配权值来判定恶意代码及其变种.

4 行为分析

行为分析是恶意代码变种识别的基础环节，本文采用动态污点分析技术实现对恶意代码行为的细粒度分析.

本文对传统污点分析方法进行了扩展，加入对污点的回溯分析流程，以判断当前污点所关联的污点源，获取污点操作指令及行为和污点源之间的数据依赖关系.

本文的可回溯动态污点分析主要包括污点源标记、污点传播计算、污点回溯和污点传播结束判定.本文将敏感 API 调用的传出参数及返回值标记为污点，定义污点源的敏感 API 调用集合 *TaintSource*，该集合包含进程、线程、注册表、网络和文件五类 API 调用.在恶意代码行为监控过程中，识别其执行的 API 调用 F_i，若 $F_i \in TaintSource$，则将其传出参数及返回值标记为污点.

为进行污点传播计算，需记录污点数据状态并定义污点传播规则.本文通过构建影子内存记录污点数据状态，当监控到数据操作时，根据操作数的地址查询影子内存，判断该操作是否为污点操作.污点传播规则根据 API 调用和其他指令分别定义.由于 Windows 系统中 API 调用的多样性，难以使用统一的方式进行处理，因此需针对每个 API 调用编写污点处理规则.本文将 API 调用抽象为传入参数和传出参数及返回值两部分，即 $F=(Parameter_{in}, Parameter_{out})$，$Parameter_{in}$代表传入参数集合，$Parameter_{out}$表示传出参数及返回值集合.调用发生时，判断其传入参数是否包含污点数据，若 $Parameter_{in} \in Taint$，则进行污点传播计算，并标记 $Parameter_{out} \in Taint$，为其创建影子内存并添加到污点记录中.对于其他指令，本文将其分为数据转移指令、运算指令、比较指令、控制流转移指令四类，针对每一类编写污点传播规则.

污点回溯过程中，为判断当前污点所关联的污点源，根据当前污点记录，依次查找回边进行回溯，通过已保存的污点记录获取恶意代码行为及指令之间的关系.

在污点分析过程中，需判断代码执行的 API 调用之间的数据依赖关系和控制依赖关系.根据污点传播规则，若 API 调用 F_1 产生的污点传播至 F_2 的传入参数，则 F_1 与 F_2 之间存在数据依赖关系.控制依赖则表现为污点数据对控制流转移方向的影响，该影响通过标志寄存器影响控制流转移方向实现.本文标记标志寄存器为污点，在系统中结合反汇编引擎，当依赖于污点标志寄存器的控制流转移指令执行时，对当前指令和其后指令进行反汇编，计算后必经节点判断当前控制流转移指令的控制范围[16].如果 API 调用 F_2 在 F_1 的

控制范围之内,则 F_1 与 F_2 之间存在控制依赖关系.

污点分析结束包括:污点漂白、进程退出和执行超时.污点漂白是指污点数据被非污点数据改写;进程退出时会销毁内部的数据记录;由于一些木马和僵尸程序内部往往含有接收指令、分析的死循环,为了实现对此类恶意代码的分析,本文设置了超时时间 T,当恶意代码执行时间超过 T 时,终止污点分析过程.

5　行为特征构建

根据行为分析的结果,本文使用恶意代码执行的关键 API 调用和 API 调用之间的依赖关系,以图的形式构建恶意代码行为特征.为了提高对常用混淆技术的抗干扰能力,本文对行为特征进行了抗混淆处理.

5.1　行为特征图

根据污点分析过程提取的关键行为数据,构建恶意代码行为特征图.行为特征图 G 描述为:

$$G=(V_E,V,DE,Ins)$$

其中 V_E 表示图的入口节点,V 为其他节点,DE 表示依赖边,Ins 为相关指令记录.

行为特征图的构建包括节点的添加、依赖边的添加.节点的添加在发生关键 API 调用时进行.首先以首个敏感 API 调用作为入口节点创建行为特征图,然后动态分析每条指令,当发生 API 调用时,解析其传入参数,若包含污点数据,则将该调用作为一个新节点 V_i 添加到图中.

依赖边的添加包括数据依赖边的添加和依赖关系恢复.数据依赖边的添加与节点的添加同步进行,当新节点 V_i 添加到图中时,回溯污点传播流程,查找 V_i 与其他 API 调用节点之间的数据依赖关系,并添加依赖边.

恶意代码为干扰分析,会在代码中使用某些条件控制语句切断数据依赖关系.例如:

```
switch(x)
{case m,y=m;case n,break;…}
```

在此例中,y 传递了 x 的值,但由于使用了条件控制语句 switch,在污点分析中 y 与 x 之间不再有数据依赖关系,应有的依赖关系被切断.针对此类情况,本文在控制依赖发生时,判断其是否切断了数据依赖关系,并将被切断的依赖关系恢复.如此例,在污点分析中识别污点控制的控制流转移指令时,回溯执行记录,记录前一条影响标志寄存器指令源操作数的值 *value*.当控制依赖范围确定后,若 EIP 位于控制依赖范围内时,判定指令执行的源操作数是否与记录的 *value* 值相等,若相等,则认为是数据依赖被控制依赖切断的情况,恢复相应的依赖边.需要处理的特殊情况是,某些编译器并不统一使用 cmp 指令来实现比较操作,而是根据情况对代码进行优化,如比较值是否为 -1 时,采用的指令并非 cmp eax,-1,je xxxx;而是 inc eax,jz xxxx.对于此类情况,需要对指令集中可能引起标志位改变的代码进行全面分析.

行为特征图构建的结束条件包括:代码执行结束、全部污点被漂白或分析超时.

5.2　抗混淆处理

为降低混淆技术的干扰,提高特征的适应能力,本文对行为特征图进行抗混淆处理.

本文通过动态行为分析构建特征,因此只需对行为层混淆方法进行处理.恶意代码常用的行为层混淆方法包括:垃圾调用插入、循环变换、等价调用替换以及随机文件名等,本文即从这四个方面对进行处理.

垃圾调用是指恶意代码中与恶意行为无关的 API 调用,这些调用的存在会干扰特征匹配.本文通过判断 API 调用是否使系统状态产生改变来判定垃圾调用.从污点分析的角度,预定义可以使系统状态改变的输出集合 *OutputChange*.对于污点传播路径 $L=\{v_1,v_2,v_3,\cdots\}$(v_i 表示路径中的行为节点,$v_1\in Taint$),垃圾调用的判定及处理方法是:当 L 中的首节点 v_1 只有一条出边时,(1)若 L 中仅存在一个节点 v_1,即 v_i 满足 $Succ(v_1)=v_i$($Succ(v_1)$ 表示 v_1 的后继节点),则将 v_1 及其相关的依赖边删除;(2)若 L 中没有任何节点改变系统状态,即 $\forall v_i\in L$,v_i 满足 $v_i\in OutputChange$,则将 L 中的节点及相关的依赖边删除.

恶意代码常通过创建循环或更改循环操作粒度的方式改变特征干扰匹配.本文采用 Sreedhar 等[16]提出的循环识别算法识别代码中的循环.在污点传播过程中,为每次污点调用操作记录污点的操作范围信息.根据污点分析提取的指令记录,如果在一次循环过程中,污点内存连续且对内存的操作指令相同,则判断为循环,将循环操作的多次冗余记录缩减为一次执行信息.

等价调用是指功能相同而名称不同的 API 调用或序列,通过等价调用序列的相互替换,恶意代码可改变特征描述使匹配失效.本文通过静态分析和动态分析的经验定义等价调用库,库中包含一系列 API 调用序列的集合,每个集合中是行为等价的调用序列,并将每个集合以统一的集合节点 $v_{Set}\in V_{Set}$ 表示,v_{Set} 中保存到此集合的链接.在特征中如果出现等价调用库中某个集合中的序列,则将其替换为相应的集合节点.

恶意代码中常通过随机生成文件名或目录遍历搜索,对其他文件进行感染或从文件中搜寻所需内容.以特定文件名称为污点进行匹配的方法面对此类情况时存在局限.为此,本文定义污点文件符号,将随机文件标记为 R_{rand},遍历系统目录或特定空间获取的文件标

记为 T_{rand},并针对这两种文件类型定义匹配规则.对于随机生成文件名的情况,经分析,恶意代码普遍使用 *GetTickCount* 获取系统时间作为生成随机数的种子,生成随机数经变换后作为文件名.在污点分析中表现为以 *GetTickCount* 的返回值为污点,传播至 *CreateFile* 的过程,此时将 *CreateFile* 传入参数的文件名标记为 R_{rand}.对于遍历搜索获取文件名的情况,实现遍历需要调用 *FindFirstFile* 和 *FindNextFile*,为此标记这两个调用为污点,当污点传播至 *CreateFile* 时,标记其文件名为 T_{rand}.

抗混淆处理后,恶意代码行为特征图 G 描述为:

$$G=(V_E,V_N,V_{Set},DE)$$

其中 V_E 为入口节点,V_N 表示普通节点,V_{Set} 为集合节点,DE 代表依赖边.

6 恶意代码变种识别

6.1 特征库优化

为了简化匹配算法、提高效率并缩减存储复杂度,首先对特征库进行优化.优化过程分为两步:一、将行为特征图转化为特征树,二、特征树融合,构造特征融合查找树.

6.1.1 行为特征图转化为特征树

为了对特征进行融合,首先将特征图转化为特征树.为了不损失必要信息,本文通过扩充的方式实现转化.行为特征图是无环路的有向图,其节点和边均为有穷非空集合,本文通过保留所有的有向边且每条边不重复,并适当扩充顶点,得到树结构,记为:

$$T=(V_E',V_N',V_{Set}',DE')$$

在转换过程中,对节点的转换不受其属性影响,为方便说明,将节点统一以 V 表示.

特征图 G 至特征树 T 的转化原则为:(1)转化后 T 为树形式;(2) $DE'=DE$,即转化后 T 保留 G 的所有边;(3) V' 为 V 的扩充,在满足 T 为树的条件下,V' 为最小集.

转化算法为:对 V 中的所有节点,考察其入度 k (即入边数量),对所有 $k>1$ 的节点 v,采取以下步骤:(1)扩充 v 的数量至 k,形成节点 $v_1,v_2,\cdots,v_k$;(2)将 v 的每个入边依次分配至 $v_1,v_2,\cdots,v_k$;(3)将 v 的所有出边分配至 v_1;(4)将所有扩充节点标记为 *Repeat*,并增加标注记录 $Note(v)=(k,link,link_{equ})$,其中 $link$ 链接到 v_1,可满足任一条入边均能连接到出边以下的节点继续匹配,扩充节点之间以 $link_{equ}$ 链接.

扩充节点的出边分配的节点 v_1 为随机选取,因此包含扩充节点的生成树不唯一,但其节点和边的个数均相同,且扩充节点已标注到出边的链接,即在匹配中均可以转到出边,因此对于匹配结果不存在影响.经过对恶意代码样本的分析可知,在特征图中很少出现入边大于 1 的情况,因此,本文的转化方式对特征库的数据存储量影响很小.

转化为特征树后,为叶节点添加记录 $L(v_L)=(SigTree,VN,EN)$,其中 *SigTree* 为样本标识,*VN* 为该特征中的节点总数,其中所有扩充节点按一个节点计算,*EN* 为边总数.

6.1.2 特征树融合

转化为特征树后,将特征树融合为特征融合查找树.

比较特征库中的各特征树,将相同节点进行融合.节点融合规则是:当 API 调用名称相同时,即认为 $v_1=v_2$,融合为同一节点并保存每个分节点的参数信息以及数据记录.

首先随机从库中找出一特征树,将其作为构造特征融合查找树的基础,标记为 TM_A,遍历特征库中的其他特征树 T_N,判断其根节点 v_{EN} 与 TM_A 的根节点 v_{EA},若 $v_{EN}\neq v_{EA}$,继续遍历其他特征树,直至选出 $v_{EN}=v_{EA}$ 的特征树,将根节点融合.然后继续融合当前特征树 T_N 和 TM_A 的子树.从 v_{EA} 的子节点 v_{NA} 开始,与 v_{EN} 的子节点 v_{NN} 比较,若 $v_{NN}\neq v_{NA}$,则选择 v_{NA} 的兄弟节点与 v_{NN} 比较.若 v_{EN} 的所有子节点都无法与 v_{NN} 匹配,则将以 v_{NN} 为根的子树直接添加在 TM_A 的根节点之下.若有 $v_{NN}=v_{NA}$,则将两节点融合,继续递归比较其子树.特征树融合过程通过深度优先的遍历算法进行.

当任意特征树都已融合或在库中无法找到剩余特征树的匹配时,特征树融合完成,特征库表示为特征融合查找树森林.

6.2 匹配算法

为支持匹配过程,在特征树融合时为每个节点标记数据记录及比较域:

$$R(v)=\{FName,SigTree,Parameter,matched,visited\}$$

其中 *FName* 表示 *API* 调用名称,*SigTree* 表示节点所属的源特征树,*Parameter* 表示节点包含的所有参数,*matched* 表示节点是否已匹配,*visited* 表示该节点是否已回溯过.匹配时,为每个特征树添加节点和边的匹配计数器 *NCounter* 和 *ECounter*.

当监控的目标代码调用敏感 API 时,由特征库中查找根节点与当前 API 调用匹配的特征融合查找树 TM_1.若没有符合条件的特征树,则舍弃当前行为记录继续监控.当找到 TM_1 时,设置当前节点为 v_{cur},标记传出参数及返回值为污点,进行污点传播运算.当发生新的 API 调用 F_i 时,查找 v_{cur} 的子树中是否有根节点与 F_i 匹配,若有,则匹配依赖边,并置当前节点为 v_{cur};否则关于此敏感 API 的污点运算停止.在匹配过程中未到达叶

子节点且当前的污点传播未终止时，仅考虑节点所表示 API 的名称进行匹配.

在匹配过程中对扩充节点和集合节点需特殊处理.遇到扩充节点时，在其匹配后，判断是否存在子节点，若不存在，则根据标注的信息，查看其 *link* 链接的节点是否有子节点且是否标记为 *matched*，若有子节点且未标注为 *matched*，则由该处向下匹配，否则停止匹配此分支.遇到集合节点时，转到相应的等价调用集合中继续匹配，若集合中有一条序列匹配成功，则转回树中继续匹配，否则匹配结束.

从查找树的角度，匹配结果可分为匹配到达叶节点和在匹配路径中停止.当匹配到达叶节点时，回溯融合查找树，根据路径上节点和边的 *SigTree* 域识别其所属的源特征树，并计算相应计数器的计数值.当匹配在路径中停止时，首先比较当前行为的参数与节点的 *Parameter* 域，判定所属的源特征树，同时以该节点为起点回溯匹配路径.在比较参数时要注意符号标注参数的处理：如遇到 *CreateFile* 且文件名标记为 R_{rand} 时，则忽略比较文件名参数，只比较其他部分；当遇到 *CreateFile* 且文件名标记为 T_{rand} 时，通过 *CreateFile* 的 *lpFileName* 参数获取该文件的路径，只比较路径是否相同，忽略文件名参数.在回溯计算中为避免路径重复回溯而引起的重复计算，标记已回溯节点的 *visited* 域为 *true*，当分支回溯到 *true* 节点时停止.

匹配完成后，根据叶节点记录 $L(v_L)$ 中样本特征的总节点和边个数 *VN* 和 *EN*，结合匹配节点和边的计数值 *NCounter* 和 *ECounter*，计算匹配权值：

$$M = \frac{NCounter + ECounter}{VN + EN}$$

然后根据识别需求定义阈值 T，当 $M \geqslant T$ 时，可判断目标为恶意代码或其变种.

6.3　触发条件处理

当前的检测工具在分析恶意代码时常使用轻量级虚拟机执行代码进行启发式扫描，多数情况下，此类工具只是简单设置超时，对于加入触发条件的恶意代码，通常因为分析超时而造成漏报.

为降低恶意代码常用触发条件的影响，本文构建了触发条件处理引擎，在动态分析代码的过程中对常用触发条件进行处理.该引擎通过识别并满足执行条件使代码继续运行.恶意代码常用触发条件主要包括执行特权指令、调用 *Sleep* 函数延时、循环数学计算、等待用户输入和判断系统时间等.其中执行特权指令的方法主要用于对抗轻量级虚拟机，由于本文的系统基于硬件模拟器实现，此种方法无法干扰.

本文主要对调用 *Sleep* 函数延时、循环数学计算、等待用户输入和判断系统时间四种情况进行处理.针对调用 *Sleep* 函数延时，拦截 *Sleep* 函数，当调用发生时读取堆栈中的延时时间，并修改 *cpu_get_ticks* 系列函数，设置其返回值为相应时间以触发后续行为.针对循环数学计算，由分析得知用于干扰的循环运算往往对非关键数据进行操作，在监控时，判断非污点数据操作的指令中是否存在循环，发现循环时反汇编分析将循环控制流向前转移的指令，再次执行到该指令时，将虚拟 CPU 中的 EFLAGS 寄存器对应标志位求反，迫使循环终止以执行后继指令.对等待用户输入的情况，加入值守代码，在特定时间间隔调用硬件模拟器的 *do_send_key* 和 *do_mouse* 等系列接口函数，向虚拟系统发送鼠标和键盘消息以满足触发条件.对判断系统时间的情况，判定指令执行的回边识别循环，若循环的控制流转移指令被循环中获取系统时间的污点数据控制，则修改控制流转移指令的对应标志位实现继续执行.

7　系统实现与实验

7.1　系统实现

本文的恶意代码变种识别原型系统由虚拟系统、行为分析、特征构建和恶意代码识别四个功能模块组成，如图 1 所示.

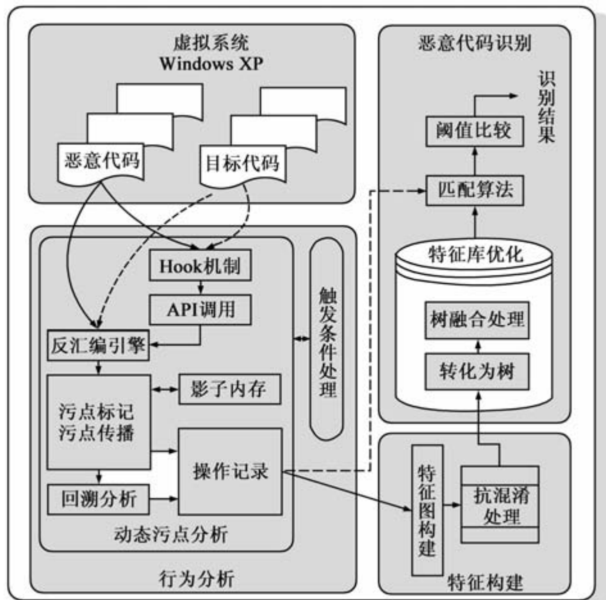

图1　原型系统示意图

本系统在硬件模拟平台上运行虚拟操作系统 Windows XP，对其中运行的进程实现细粒度监控，获取代码执行的各种信息.

行为分析部分，本文实现并改进了污点分析引擎，实现了 API 调用拦截、反汇编、影子内存及回溯分析等.API 调用拦截通过判断调用的入口地址实现，在进程加载时分析进程上下文，查找加载的动态链接库导出表，将 API 调用地址记录在调用列表中.在代码执行中比较

虚拟 CPU 的当前 EIP 和 API 调用列表的入口地址判断 API 调用,若是则读取堆栈中的参数,该方法不对系统和程序做修改,具有良好的隐蔽性和透明性.本系统整合了反汇编引擎,在分析过程中反汇编每一条执行的指令,并记录该指令执行时的 CPU 各项内容,进行污点传播运算.影子内存所映射的区域包括进程映射进内存后的文件、进程在堆中分配的内存、进程栈等,记录污点内存的相关信息包括内存所在的位置、以字节计算的长度,内存的类型,内存当前的状态、是否为寄存器以及寄存器状态等.为实现回溯分析,在污点传播记录中使用双链表结构,通过回边连接当前污点操作指令和污点源.为辅助分析,本系统构建了触发条件处理引擎,通过满足目标代码执行条件,挖掘其完整行为.

特征构建部分,利用在行为分析部分获取的关键 API 调用及依赖关系,构建行为特征图,在此部分中实现了抗混淆处理引擎,对行为特征图进行抗混淆处理.

恶意代码识别模块中,实现了恶意代码特征库的优化,通过匹配算法将目标代码的行为监控数据与特征进行匹配,计算匹配权值与用户设定的阈值比较,得到识别结果.

7.2 实验

为了验证恶意代码变种识别系统的实际效果,由网络病毒数据库 VXHeavens[17]选取了真实恶意代码样本与手工加壳及混淆样本进行了实验.实验结果表明,本系统对恶意代码及其变种具有良好的识别能力.

下面以 SDBot、Bagle 及 NetSky 样本为例说明实验结果.实验中的阈值设为 0.7.

(1)SDBot

SDBot 是一种基于 IRC 协议蠕虫病毒.本实验从 VXHeavens 数据库中获取后缀为.b、.bx和.by的样本,通过手工方法处理得到后缀为.up、.ex、.as和.ma的样本.

由 SDBot.b 样本提取特征作为基础,实验结果表 1 和表 2 所示.

表 1 SDBot 变种样本实验结果

样本名称	节点匹配数	边匹配数	匹配权值
SDBot.b*	26	30	1
SDBot.up	26	30	1
SDBot.ex	22	24	0.82
SDBot.as	20	26	0.82
SDBot.ma	22	26	0.86
SDBot.bx	22	28	0.93
SDBot.by	21	25	0.82

在实验中注意到,Asprotect 加壳引起了分析时间的增长,这是由于其使用了特殊尚 CPU 指令代码,该问题可通过增添及修正污点传播规则解决.SDBot 中存在协议解析和分派函数.协议解析过程以网络接收函数 WSARecv 为污点源,通过该污点源,在优化时将 6 个不同的远程指令触发行为特征融合为一个.最后产生的 3 个特征分别表示复制自身到系统目录、安装自启动项和协议解析及行为执行.在实验中发现,SDBot.bx 包含大量循环代码,这些操作在抗混淆处理的过程中被缩减,没有影响分析结果.

表 2 SDBot 变种样本分析及识别时间

样本名称	加壳类型	分析时间	识别时间
SDBot.b*	无	11m20s	6s
SDBot.up	UPX	11m51s	6s
SDBot.ex	ExeCrypt	14m04s	8s
SDBot.as	Asprotect	20m35s	10s
SDBot.ma	手工混淆	14m46s	7s
SDBot.bx	UPX	12m07s	6s
SDBot.by	ASPack	13m12s	8s

(2)Bagle

Bagle 是一个电子邮件蠕虫病毒. Bagle 特征由 Bagle.a 和 Bagle.e 分别提取,实验结果如表 3 和表 4 所示.

表 3 Bagle 变种样本实验结果

样本名称	节点匹配数	边匹配数	匹配权值
Bagle.a*	59	58	1
Bagle.b	46	46	0.79
Bagle.e*	33	30	1
Bagle.f	28	25	0.84
Bagle.g	24	21	0.71

表 4 Bagle 变种样本分析及识别时间

样本名称	加壳类型	分析时间	识别时间
Bagle.a*	无	7m55s	3s
Bagle.b	UPX	6m40s	5s
Bagle.e*	PeX	7m40s	6s
Bagle.f	PeX	6m35s	5s
Bagle.g	PeX	6m03s	8s

在实验中发现,Bagle 使用 GetTickCount 获取系统时间,运算后作为随机数种子,用于生成邮件地址信息发送病毒邮件.在特征生成的过程中本文进行了符号标注,从而实现了正确匹配.Bagle 执行时包含如下时间条件处理:使用 *GetLocalTime* 获取当前系统时间作为触发条件;初始化运行完成后,调用 *Sleep* 函数进入休眠状态,每秒钟调度自身执行一次;创建线程每隔十分钟向硬编码的网络地址发送被感染机器信息,时间间隔通过 *Sleep* 函数实现.对于获取当前时间判断是否执行的问题,实验中通过手工干预解决,其余调用 Sleep 函数的问题由触发条件处理引擎自动处理,由此获取了样本的完整行为并得到正确的识别结果.

(3)Netsky

NetSky 是一种通过电子邮件传播的 Win32 蠕虫.实验中由 NetSky.ad 提取行为特征识别变种,结果如表 5

和表 6 所示.

表 5 Netsky 样本实验结果

样本名称	节点匹配数	边匹配数	匹配权值
NetSky.ad*	17	24	1
NetSky.aa	13	18	0.76
NetSky.af	15	22	0.90
NetSky.c	13	20	0.80
NetSky.r	15	20	0.85
NetSky.t	14	22	0.88

表 6 Netsky 样本分析及识别时间

样本名称	加壳类型	分析时间	识别时间
NetSky.ad*	PCPEC	8m32s	7s
NetSky.aa	ASPack	14m48s	9s
NetSky.af	PCPEC	12m07s	6s
NetSky.c	UPX	12m24s	7s
NetSky.r	PEtite	12m35s	9s
NetSky.t	UPX	10m23s	8s

由分析获知,NetSky 在首次感染时向操作系统的关键目录复制自身;使用 *FindFileFirst* 和 *FindFileNext* 遍历硬盘目录,将自身拷贝到名称中带有 *share* 字符的目录下;并修改注册表键值实现自启动.在实验中 NetSky.ad 样本复制自身时包含大量循环,由于本文对循环进行了缩减,该行为对本文的识别方法没有影响.

(4)误报率评估

使用本文的识别系统对 4 种正常程序进行测试,以检验系统的误报率,实验结果如表 7 所示.由实验数据可知,正常程序与恶意代码行为特征的匹配权值均远低于阈值 0.7.由此可见,本系统能较好地区分恶意代码与正常代码.

表 7 正常代码样本实验结果

特征样本	SDBot.b*			Bagle.a*			NetSky.ad*		
匹配数样本名称	节点	边	权值	节点	边	权值	节点	边	权值
Notepad.exe	4	5	0.16	6	7	0.11	4	5	0.22
Calc.exe	1	2	0.05	1	2	0.03	0	0	0
Iexplore.exe	8	9	0.30	10	12	0.19	6	7	0.32
Ftpserv.exe	7	9	0.29	7	8	0.13	5	8	0.32

8 结论

本文提出了一种抗混淆的恶意代码变种识别系统,该系统包括行为分析、特征构建及恶意代码变种识别三个功能部分.首先使用可回溯的动态污点分析技术跟踪并记录代码执行的关键 API 调用、依赖关系以及指令信息;然后利用这些行为数据构建行为特征,并以图的形式进行描述;在识别部分,首先实现特征库的优化,然后对目标代码进行监控,配合触发条件处理引擎,分析目标代码关键行为,与特征进行匹配计算匹配权值,得到识别结果.通过对多个恶意代码及变种样本的实验,验证了本系统对混淆产生的恶意代码变种的分析及识别能力.

在下一步工作中,将在系统中整合多路径特征挖掘功能,并对目标代码的匹配过程进行优化.

参考文献

[1] J Ferrante, K J Ottenstein, J D Warren. The program dependence graph and its use in optimization[J]. ACM Transactions on Programming Languages and Systems, 1987, 9(3): 319 – 349.

[2] J Newsome, D Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[A]. In Proc. of the 12th Annual Network and Distributed System Security Symposium (NDSS)[C]. 2005.

[3] H Yin, D Song, M Egele, C Kruegel, E Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis[A]. 14th ACM Conference on Computer and Communications Security, Alexandria, VA, November 2007.

[4] M Christodorescu, S Jha, C Kruegel. Mining specifications of malicious behavior[A]. In Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)[C]. 2007.

[5] M Egele, C Kruegel, E Kirda, H Yin, D Song. Dynamic Spyware Analysis[A]. In Proceedings of the 2007 Usenix Annual Conference (Usenix'07)[C]. 2007.

[6] G Jacob, H Debar, E Fillol. Behavioral detection of malware: from a survey towards an established taxonomy[J]. Journal in Computer Virology, 2008, 4(3): 251 – 266.

[7] 彭宏,王军.基于支持向量机的病毒程序检测方法[J].电子学报,2005,33(2):276 – 278.

[8] C Kolbitsch, P M Comparetti, C Kruegel, E Kirda, X Zhou, X Wang. Effective and efficient malware detection at the end host [A]. In USENIX Security Symposium, 2009.

[9] G Bonfante, M Kaczmarek, J Y Marion. Architecture of a morphological malware detector[J]. Journal in Computer Virology, 2008, 5(3): 263 – 270

[10] V S Sathyanarayan, P Kohli, B Bruhadeshwar. Signature Generation and Detection of Malware Families[A]. LNCS, In Proceedings of the 13th Austalasian conference on Information Security and Privacy[C]. 2008, Vol. 5107: 336 – 349.

[11] F Bellard. Qemu, a fast and portable dynamic translator[A]. In USENIX Annual Technical Conference, FREENIX Track, 2005.

[12] U Bayer, P Milani Comparetti, C Hlauschek, C Kruegel, E Kirda. Scalable, Behavior-Based Malware Clustering[A]. In Network and Distributed System Security Symposium (NDSS), 2009.

[13] 王祥根，司端锋，冯登国，苏璞睿．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．

[14] D Brumley，C Hartwig，Z Liang，J Newsome，P Poosankam，D Song，H Yin．Automatically identifying trigger-based behavior in malware[A]．Botnet Analysis and Defense[M]，Springer，2008．65－88．

[15] J R Crandall，G Wassermann，D A S de Oliveira，Z Su，S F Wu，F T Chong．Temporal search：Detecting hidden malware timebombs with virtual machines[A]．In Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)[C]．2006，34(5)：25－36．

[16] V C Sreedhar，G R Gao，Y F Lee．Identifying loops using DJ graphs[J]．ACM Transactions on Programming Languages and Systems (TOPLAS)，1996，18(6)：649－658

[17] VX Heavens[OL]．http://www.netlux.org．

作者简介

王 蕊 女，1981 年出生于黑龙江哈尔滨，博士研究生，主要研究领域为恶意代码分析与防范．

E-mail：wangrui@is.iscas.ac.cn

苏璞睿 男，1976 年出生于湖北宜昌，副研究员，硕士生导师，主要研究领域为恶意代码分析与防范．

E-mail：supurui@is.iscas.ac.cn

杨 轶 男，1982 年出生于河南鹤壁，博士研究生，主要研究领域为恶意代码分析与防范．

E-mail：FirefoxXP@is.iscas.ac.cn

冯登国 男，1965 年出生于陕西榆林，研究员，博士生导师，主要研究领域为密码学与信息安全．

E-mail：Feng@is.iscas.ac.cn

（上接第 2321 页）

[8] Jang Gil-Jin；Lee Te-Won．Single-channel signal separation using time-domain basis functions[J]．IEEE signal processing letters，2003，10(6)：168－171．

[9] 成谢锋，陶冶薇，张少白，等．独立子波函数和小波分析在单路含噪信号中的应用研究：模型与关键技术[J]．电子学报，2009，(37)：1522－1528．
Cheng Xie-Feng．Applications of independent sub-band functions and wavelet analysis in single-channel noisy signal BSS：Model and crucial technique[J]．Acta Electronica Sinica，2009，(37)：1522－1528．(in Chinese)

[10] Wold S．Cross valedictory estimation of the number of components in[J]．Techno metrics，1978，20：397－406．

[11] Qin S J，Dunia R．Determining the number of principal components for best reconstruction[J]．Journal of Process Control，2006，(10)：245－250．