MACSPMD:基于恶意API调用序列模式挖掘的恶意代码检测

荣俸萍[1] 方 勇[2] 左 政[1] 刘 亮[2]
(四川大学电子信息学院 成都 610065)[1] (四川大学网络空间安全学院 成都 610065)[2]

摘 要 基于动态分析的恶意代码检测方法由于能有效对抗恶意代码的多态和代码混淆技术,而且可以检测新的未知恶意代码等,因此得到了研究者的青睐。在这种情况下,恶意代码的编写者通过在恶意代码中嵌入大量反检测功能来逃避现有恶意代码动态检测方法的检测。针对该问题,提出了基于恶意API调用序列模式挖掘的恶意代码检测方法MACSPMD。首先,使用真机模拟恶意代码的实际运行环境来获取文件的动态API调用序列;其次,引入面向目标关联挖掘的概念,以挖掘出能够代表潜在恶意行为模式的恶意API调用序列模式;最后,将挖掘到的恶意API调用序列模式作为异常行为特征进行恶意代码的检测。基于真实数据集的实验结果表明,MACSPMD对未知和逃避型恶意代码进行检测的准确率分别达到了94.55%和97.73%,比其他基于API调用数据的恶意代码检测方法的准确率分别提高了2.47%和2.66%,且挖掘过程消耗的时间更少。因此,MACSPMD能有效检测包括逃避型在内的已知和未知恶意代码。

关键词 恶意代码检测,逃避型恶意代码,序列模式挖掘,API调用序列,分类
中图法分类号 TP309.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.05.022

MACSPMD:Malicious API Call Sequential Pattern Mining Based Malware Detection

RONG Feng-ping[1] FANG Yong[2] ZUO Zheng[1] LIU Liang[2]
(College of Electronic Information,Sichuan University,Chengdu 610065,China)[1]
(College of Cybersecurity,Sichuan University,Chengdu 610065,China)[2]

Abstract Researchers give preference to dynamic analysis based malware detection methods with capability of nullifying the effects of polymorphism and obfuscation on malware and detecting new and unseen malwares. In this case,malware authors embed numerous anti-detection functions in to malware to evade the detection of existing dynamic malware detection methods. To solve this problem,a malware detection method MACSPMD based on malicious API call sequential pattern mining was proposed. Firstly,dynamic API call sequences of the files are gotten by real machine which simulates the actual running environment of the malware. Secondly,the malicious API call sequence patterns that can represent the potential malicious behavior patterns are mined by introducing the concept of objective-oriented association mining. Finally,the malicious API call sequences are used as abnormal behavior feature to detect malware. The experimental results based on real data set show that MACSPMD achieves 94.55% and 97.73% of detection accuracy on unknown and evasive malware respectively. Compared with other malware detection methods based on API call data,the detection accuracy of unknown and evasive malware is improved by 2.47% and 2.66% respectively,and the time consumed in the mining process is less. MACSPMD can effectively detect known and unknown malware,including escape type.

Keywords Malware detection,Evasive-malware,Sequetial pattern mining,API call sequence,Classification

1 引言

恶意代码是由恶意攻击者设计的,其是在未经用户许可的情况下在用户计算机系统或其他终端上安装运行以实现某种恶意目的的软件。在过去十年里,随着信息技术的迅猛发展,恶意代码对人们的正常工作和生活造成了巨大的威胁,给经济的发展带来了一系列重大损失和不利影响。恶意软件检测也因此成为计算机安全领域最关键的问题之一。

目前,杀毒软件是一种成熟的恶意代码检测和对抗软件,这些杀毒软件主要使用基于特征码的检测方法,即从已知恶意代码中提取能唯一标识此恶意代码的特征码,在进行检测时使用提取的特征码来识别已知恶意代码。特征码检测技术由于具有方法简单、检测速度快、误报率低的优点,在业界得到了广泛应用。然而,特征码检测技术也有很大的局限性,它只能检测已知的恶意代码,对使用代码混淆和多态技术[1-4]修改的已知恶意代码和新出现的未知恶意代码却无能为力。

到稿日期:2017-07-19 返修日期:2017-10-06

荣俸萍(1993-),女,硕士生,CCF学生会员,主要研究方向为恶意代码检测、机器学习;**方 勇**(1966-),男,博士,教授,主要研究方向为信息安全、网络信息对抗;**左 政**(1986-),男,博士生,主要研究方向为恶意代码检测、内核安全,E-mail:leftzheng@gmail.com(通信作者);**刘 亮**(1982-),男,博士,讲师,主要研究方向为网络信息与系统安全。

由于恶意代码动态分析技术能有效对抗恶意代码的多态和代码混淆技术,而且能检测新的未知恶意代码,因此本文采用动态分析技术进行恶意代码的检测。恶意代码动态分析的标准过程是在一个安全、透明和独立的分析环境里运行可移植执行(Portable Executable,PE)文件,并对其动态行为进行监控。这些分析环境依靠各种沙箱、虚拟化和仿真技术而建立,用来模仿文件真正的运行时环境[5]。由于分析环境和真机系统在 CPU 语义和指令执行时间等方面存在固有差异,因此完全地模仿真机系统是不可能的。为了对抗这种动态分析技术,逃避型的恶意代码利用上述差异来判断自己是否运行在分析环境中,如果自己处于分析环境,逃避型恶意代码会隐藏自己的恶意行为并呈现出终止执行的状态,同时模仿一个正常软件的行为或企图破坏分析系统等。

为了改进现有恶意代码动态分析技术对逃避型恶意代码检测的不足,本文直接使用真机对恶意代码进行动态分析。在动态分析的基础上,本文通过结合广义序贯模式挖掘算法(Generalized Sequential Pattern,GSP)[6]和面向目标的关联挖掘技术(Objective-Oriented Association,OOA)[7],提出了一种恶意序列模式挖掘算法,并将挖掘到的恶意序列模式作为特征,使用机器学习技术来自动化检测已知、未知和逃避型恶意代码。

2 相关工作

现有的利用数据挖掘和机器学习技术来自动化检测已知和未知恶意代码的智能恶意代码检测方法主要分为两个阶段:特征提取和检测。在第一个阶段,从样本信息中挖掘出能区分恶意和正常样本的特殊模式,并将其作为特征来表征样本文件。在第二个阶段,基于提取的特征,应用分类技术实现恶意代码的自动检测。根据分析技术的不同,将这些智能恶意代码检测方法提取的特征分为静态特征和动态特征两种。

2.1 基于静态特征的智能恶意代码检测方法

静态特征主要包括从可移植执行(PE)文件和其反汇编文件中提取的字节码、汇编指令、导入函数和分节信息等特征。例如,文献[8]从文件中提取单字节频率、4-gram 字节码、汇编指令、导入函数和组合汇编指令序列 5 种不同的静态特征,然后使用支持向量机和决策树算法建立分类模型进行恶意代码检测。Ye 等[9]开发了一款智能恶意代码检测系统(IMDS),它首先从每个可移植执行样本中获取导入函数信息,然后使用面向目标的关联挖掘算法来生成面向目标的关联规则,最后使用基于关联规则的分类方法检测恶意代码。Fan 等[10]于 2016 年基于汇编指令序列,使用序列挖掘技术来挖掘恶意指令序列模式并将其作为特征,再使用改进的近邻算法建立分类模型进行恶意代码的自动化检测。但是,使用基于静态特征的智能恶意代码检测方法会受到恶意代码的多态、代码混淆、加密和加壳等技术的影响。

2.2 基于动态特征的智能恶意代码检测方法

动态特征主要来源于 PE 文件动态运行时与操作系统交互产生的动态行为信息,主要包括文件、注册表、进程、网络、应用程序接口调用等方面的特征。因此,基于动态特征的智能恶意代码检测方法能够更好地检测与已知恶意代码样本行为相似的未知恶意代码,而不受恶意代码的多态、代码混淆、加密和加壳等技术的影响。由于可移植执行文件对应用程序接口(Application Programming Interface,API)的调用能够反映出文件的行为信息,因此作为基于动态特征的智能恶意代码检测方法使用的最有效的特征之一,API 调用相关特征被研究者广泛使用。Ahmadi 等[11]于 2013 年使用迭代模式挖掘的方法来提取 API 频繁迭代模式,然后计算其费舍尔得分进行特征选择。这种方法能够得到较高的检测率和较低的误报率。韩兰胜等[12]使用 API 函数名、输入参数以及它们的组合信息 3 类特征,并通过信息增益选择对恶意软件和正常软件识别率高的特征进行恶意代码的检测。李盟等[13]于 2015 年基于文件 API 调用序列,改进传统 n-gram 模型,添加 n-gram 频次信息以及各 API 间的依赖关系,构建改进的 n-gram 模型进行特征的选取,最后利用多种机器学习算法对样本进行分类检测。大部分基于动态特征的恶意代码检测方法不能有效地检测逃避型恶意代码。

目前,基于动态特征的智能恶意代码检测方法使用各种沙箱、虚拟化和仿真技术作为动态运行文件的分析环境。为了对抗这种动态分析技术,逃避型恶意样本会对自身的运行环境进行检测,如果自身运行在分析环境中,其会隐藏自己的恶意行为并呈现出终止执行的状态,同时模仿一个正常的软件行为或企图破坏分析系统等。因此,一种自动化检测逃避型恶意代码的思路是在多个分析环境中执行同一个文件,然后比较其行为,并且认为行为偏差是被检测文件试图逃避一个或多个分析系统的证据。例如,Balzarotti 等[14]在对比系统中执行文件并记录其系统调用序列和参数,然后在分析系统中将之前记录的参数作为系统调用的参数重放文件的执行过程,当文件表现出不同于对比系统中所得到的系统调用跟踪记录时,判定该文件为逃避型恶意代码。Kirat 等[15]提取了同一文件在真机、虚拟化、仿真等不同环境中运行产生的行为轮廓文件,然后通过计算行为轮廓文件之间的层级相似性来自动化检测逃避型恶意软件。上述方法都需要在多个分析环境中执行同一个文件,而 Naval 等[16]收集了文件在硬件虚拟化平台 Ether[17]中执行产生的系统调用序列,并使用了相邻系统调用间的转换概率作为特征来表示恶意代码的恶意行为、环境感知行为和感知到分析环境后的不正常行为,从而检测恶意代码并判断具体的逃避类型。这种方法虽然对包括逃避型在内的恶意代码有不错的检测效果,但是只使用相邻系统调用这种 2-gram 序列模式特征并不能有力地表示上述各种行为模式,而且这种方法对部分能够检测到 Ether 并且阻止 Ether 生成系统调用序列的逃避型恶意代码无能为力。

本文提出了一种基于恶意 API 调用序列模式挖掘的恶意代码检测方法(MACSPMD),其能自动化地检测包括逃避型在内的已知和未知恶意代码。针对程序动态运行时得到的 API 调用序列数据,这种方法使用改进的序列挖掘算法来发现恶意 API 调用序列模式,并将挖掘到的恶意 API 调用序列模式作为异常行为特征进行恶意代码的检测。本文的主要贡献如下。

(1)提出有效的恶意序列模式挖掘算法:本文提出的序列挖掘算法在挖掘的过程中引入了 OOA 的概念,以便挖掘出能够检测未知恶意样本的恶意 API 调用序列模式。与此同时,本文在挖掘算法中设置了剪枝规则来过滤冗余序列模式,以减少序列模式挖掘过程中的时间和空间消耗。

(2)使用真机环境进行恶意代码的动态分析：本文使用真机分析环境获取文件在动态运行过程中调用的真实且完整的 API 调用序列数据，避免出现逃避型恶意样本检测到分析环境后不触发恶意行为的情况。

(3)通过单次执行文件检测逃避型行为：本文通过在单一的分析环境下单次执行文件得到的行为信息即可有效检测逃避型恶意软件，避免了在不同分析环境多次执行同一文件带来的时间和设备资源的消耗。

(4)进行充分的实验研究，证明了方案的有效性：本文收集了大量真实的恶意和正常 PE 文件，并使用这些文件进行一系列的实验来评估所提出的恶意代码检测方法。实验表明，本文提出的恶意代码检测方法能有效地检测未知恶意代码和逃避型恶意代码。

3　设计与实现过程

本节对本文提出的恶意代码自动检测方法及其系统实现进行详细的介绍。

3.1　恶意代码自动检测系统的概述

本文提出的恶意代码自动检测系统的主要目的是有效检测出有别于正常 PE 文件的恶意代码。为了达到这一目的，通过使用随机森林算法[18]构建分类模型来进行恶意代码的检测。如图 1 所示，系统的实现流程主要分为随机森林构造和恶意代码检测两个阶段(其中虚线标示的过程是构造阶段特有的处理流程)。

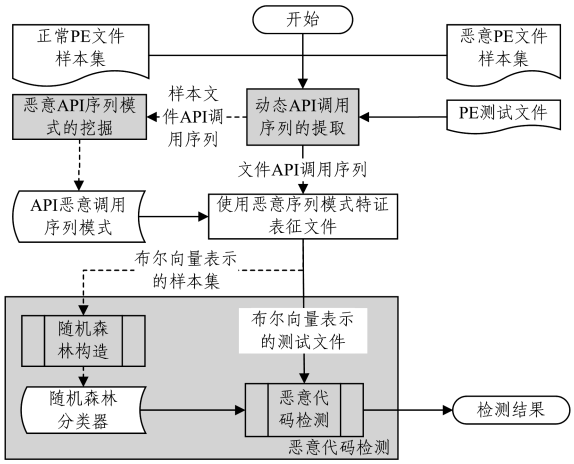

图 1　恶意代码自动检测系统的流程图

Fig. 1　Flow chart of automatic malicious code detection system

(1)随机森林构造阶段：首先，提取样本文件的动态 API 调用序列；其次，从样本文件的 API 调用序列数据中挖掘出对恶意和正常样本文件有区分作用的恶意 API 调用序列模式；然后，使用挖掘出的恶意 API 调用模式来表示这些样本文件；最后，利用布尔向量表示的样本数据集构建随机森林分类器。

(2)恶意代码检测阶段：首先，提取测试文件的动态 API 调用序列；然后，使用构造阶段挖掘出的恶意 API 调用序列模式来表示此测试文件；最后，把得到的布尔向量表示的测试数据输入到构造阶段得到的随机森林分类器中，以进行恶意代码检测。

从图 1 中可以看到，系统主要由动态 API 调用序列的提取、恶意 API 序列模式的挖掘和恶意代码检测 3 个重要部分组成。

3.2　动态 API 调用序列的提取

动态 API 调用序列的提取是一个自动化的动态分析过程，它通过在真机分析环境中动态运行和监控每个 PE 文件得到其真实且完整的 API 调用序列。

为了对抗现有的动态分析技术，逃避型的恶意样本通过对调试器、仿真机、沙箱、虚拟机以及分析进程进行检测来判断自身是否处于分析环境中，如果逃避型恶意代码检测到自己运行在分析环境中，它将隐藏自身的恶意行为而表现出其他行为，从而逃避动态分析技术的检测。传统的 Cuckoo 恶意代码分析系统[19]使用虚拟机作为动态分析环境，无法有效提取逃避型恶意代码真实、完整的 API 调用序列。本文参考虚拟分析环境侦测项目 Pafish[20]中的侦测方法来对 Cuckoo 进行相应的修改，以实现动态 API 调用序列的提取。修改后的分析系统直接使用真机作为分析环境，并且在分析进程中加入模拟人与系统交互行为的功能，同时还隐藏了分析环境中的分析进程。其中，模拟人与系统交互行为的功能是通过 Python 的图形界面自动化操作库 PyAutoGUI 实现的，在分析过程中使用 PyAutoGUI 模拟出鼠标的移动、点击、拖拽、滚屏和键盘按键输入、按住操作以及鼠标和键盘的热键同时按住等一系列交互操作行为。这些改进措施使得本文的动态分析系统相对于其他分析平台(Anubis[21]，Ether[17]，CWSandbox[22]等)能更好地对抗逃避型恶意代码中包含的反检测机制，更有利于触发恶意代码的恶意行为，从而提取真实且完整的 API 调用序列，进一步达到有效检测恶意代码这一目的。

修改后的 Cuckoo 动态分析系统的网络拓扑图如图 2 所示。Cuckoo 的管理端程序运行在 Cuckoo 主机上，真机分析环境通过局域网与 Cuckoo 主机相连，管理端程序对真机分析环境进行远程管理：分发被分析文件到真机分析环境，开始真机分析任务，提取文件的 API 调用序列，结束分析任务后发送远程过程调用(RPC)请求来重启真机分析系统并使用 Clonezilla[23]快速还原真机分析系统到干净的分析环境。Clonezilla 对系统进行备份还原时只对硬盘中使用的区块进行保存和恢复，因此相对于同类工具(Acronis True Image[24]，Norton Ghost[25]等)，Clonezilla 极大地提升了克隆的效率。在真机分析系统的实际还原过程中，Clonezilla 的还原速率达到了 9 GB/min。

此外，Cuckoo 客户端程序被安装并运行在真机分析环境中，其主要功能是启动被分析文件并对其 API 调用进行监控和记录，在分析完成后将监控结果传回 Cuckoo 主机。

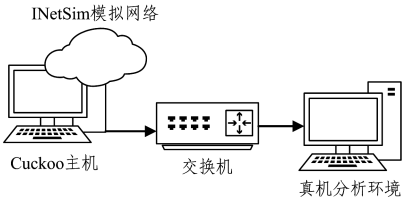

图 2　修改后的 Cuckoo 动态分析系统的网络拓扑图

Fig. 2　Network topology of modified Cuckoo dynamic analysis system

此外，为了把真机分析环境伪装成一个正常的办公电脑，在真机分析环境中安装了必要的程序运行环境、办公软件、常用软件、常用浏览器，并且留下这些软件的使用痕迹，如保存浏览器的历史访问记录、网站的登录凭据和使用办公软件打开用户文档的记录；同时，还在 Cuckoo 主机上使用 InetSim[26] 构造虚拟网络，以为真机分析机模拟真实的因特网环境，使得执行的文件需要访问网络才能正常运行。这些举措能够让我们观察到一些在只安装有操作系统的分析环境中不能观察到的恶意行为，如下载网络文件、获取网站登录凭据等。

在得到 PE 文件的 API 调用序列后，对这些数据进行下一步处理，以挖掘出对恶意样本和正常样本文件有区分作用的恶意 API 调用序列模式。

3.3 恶意 API 序列模式的挖掘

3.3.1 选择关键 API 调用

由于并不是所有的 API 调用都对恶意代码检测和恶意序列模式挖掘具有关键作用，因此需要从以上步骤得到的样本文件 API 调用序列中选取有利于挖掘出潜在恶意序列模式的关键 API 调用。

有利于恶意代码检测和恶意序列模式挖掘的关键 API 调用与恶意类别的可执行文件有较高的相关性。为了选取出这些关键 API 调用，通过引入相关性的概念对一个指定的 API 调用与文件类别的相关性进行定义。

定义 1 令 a_i 表示所有样本文件 API 调用序列中所有 API 调用组成的集合 A 中的第 i 个 API 调用，f_t 表示文件类别集合 F 中的第 t 个类别，定义其相关性如下：

$$Relevance(a_i,f_t)=\begin{cases}\dfrac{freq(a_i,f_t)}{\sum_{f_t\in F}freq(a_i,f_t)}, & freq(a_i,f_t)\neq 0\\ 0, & freq(a_i,f_t)=0\end{cases} \tag{1}$$

其中，$freq(a_i,f_t)$ 代表 API 调用 a_i 在文件类别 f_t 中出现的加权频率，其计算方法如下：

$$freq(a_i,f_t)=\frac{NS(a_i,f_t)}{NS(f_t)}\times\frac{NA(a_i,f_t)}{NA(f_t)} \tag{2}$$

其中，$NS(a_i,f_t)$ 和 $NS(f_t)$ 分别为包含 API 调用 a_i 的类别为 f_t 的样本文件数量和类别为 f_t 的样本文件总数，$NA(a_i,f_t)$ 和 $NA(f_t)$ 分别为 API 调用 a_i 在类别为 f_t 的样本文件 API 调用序列中出现的次数和类别为 f_t 的样本文件 API 调用序列中 API 调用的总次数。

根据定义 1，可以计算出每个 API 调用与恶意类别 M 的相关性；基于此，通过条件 $Relevance(a_i,f_M)>r$ 来选择与恶意代码相关度高的 API 调用，其中 r 为最小相关性值。最佳的 r 值选出的 API 调用集合应该能表示所有的恶意样本以及尽可能少的正常样本。通常通过实验来确定最佳的 r 值，并对选出的 API 调用编号，从而形成关键 API 调用集合。

下面使用关键 API 调用集合来进行恶意序列模式的挖掘，以避免无关 API 调用在恶意序列模式挖掘过程中产生大量冗余且无益的 API 调用序列模式，减少时间和空间消耗。

3.3.2 恶意 API 调用序列模式挖掘

恶意序列模式挖掘的主要目的是挖掘出对恶意样本和正常样本文件有区分能力的恶意 API 调用序列模式。

本文提出的恶意序列模式挖掘算法是结合 GSP 算法和 OOA 技术形成的一个改进版本。GSP 是一个通用且有效的序列模式挖掘算法，它采用对冗余候选模式的剪枝策略（候选序列模式的子序列必定是序列模式）和特殊的数据结构（实现候选模式快速访存的哈希树）来降低计算的复杂度。但是直接应用 GSP 时，它只能挖掘出恶意样本和正常样本共有的 API 调用序列模式，即直接应用 GSP 不能达到挖掘恶意 API 调用序列模式和检测恶意代码的目的。本文在 GSP 的基础上进行改进和扩展，引入 OOA 来对具有特定文件类别属性的 API 调用序列模式进行挖掘。下面对本文提出的恶意序列模式挖掘算法中涉及到的支持度和置信度进行定义。

定义 2 令 I 表示由关键 API 调用组成的 API 调用序列模式，f_M 表示文件类别集合 F 中的恶意文件类别，则 $I\rightarrow f_M(s\%,c\%)$ 是恶意模式挖掘过程中产生的一条关联规则，它表示 I 关于恶意类别 f_M 的支持度和置信度分别为 $s\%$ 和 $c\%$，其计算公式如下：

$$s\%=\frac{count(\{I,f_M\},DB)}{count(\{f_M\},DB)}\times 100\% \tag{3}$$

$$c\%=\frac{count(\{I,f_M\},DB)}{count(\{I\},DB)}\times 100\% \tag{4}$$

其中，DB 为存储样本文件 API 调用序列的样本数据库；$count(\{I,f_M\},DB)$ 为 DB 中包含序列模式 I 并且文件类别为恶意的样本文件数；$count(\{f_M\},DB)$ 为 DB 中文件类别为恶意的样本文件数；$count(\{I\},DB)$ 为 DB 中包含序列模式 I 的样本文件数。

定义 3 令 $ms\%$ 为用户定义的最小支持度，I 表示由关键 API 调用组成的 API 调用序列模式，如果 I 关于恶意类别 f_M 的支持度 $s\%\geqslant ms\%$，则称 I 为关于恶意文件类别的频繁 API 调用序列模式。

定义 4 令 $mc\%$ 为用户定义的最小置信度，I 表示关于恶意文件类别的频繁 API 调用序列模式（即满足定义 3），如果 I 关于恶意类别 f_M 的置信度 $c\%\geqslant mc\%$，则称 I 为恶意 API 调用序列模式。

下面使用关键 API 调用集合来生成 API 调用序列模式，然后从 DB 中挖掘出恶意 API 调用序列模式。在挖掘恶意序列模式的过程中，需要对产生的 API 调用序列模式和其重要统计数据进行保存。为此，设计了如下所示的数据结构 SEQPAT，每个 API 调用序列模式对应该数据结构的一个内部实例。

```
typedef struct {
set seqpat;//存储 API 调用序列模式
int count1;//存储 count({I},DB)
int count2;//存储 count({I,f_M},DB)
double s%;//存储其关于恶意类别的支持度
double c%;//存储其关于恶意类别的置信度
}SEQPAT
```

本文提出的恶意序列模式挖掘算法的伪代码如算法 1 所示。

算法 1 恶意序列模式挖掘算法

输入：最小支持度 ms%，最小置信度 mc%，关键 API 集合 C_1 和样本数据库 DB

输出：API 恶意调用序列模式集合 MP

```
function malSeqPatMining(ms%,mc%,C1,DB)
1. k=1
2. for r in DB
3.    if r is fM then
4.       count++      //store count({fM},DB)
5. for r in DB
6.       for I in Ck
7.          if I⊆r then
8.             I.count1++
9.             if r is fM then
10.                   I.count2++
11. Lk=Ø
12. MP=Ø
13. for I in Ck
14.    I.s%=I.count2/count
15.    if I.s% ≥ms% then
16.       if k==1 then
17.          Lk=Lk∪{I}
18.          I.c%=I.count2/I.count1
19.          if I.c% ≥mc% then
20.             MP=MP∪{I}
21.       else
22.          I.c%=I.count2/I.count1
23.          c′%=max({I′.c%|(I′⊆I)∧(I′∈Lk-1)})
24.          if I.c%>c′% then}
25.             Lk=Lk∪{I}
26.             if I.c% ≥mc% then
27.                MP=MP∪{I}
28. Ck+1=candGen(Lk)
29. if Ck+1≠Ø then
30.    k=k+1
31.    jump 5
32. return MP

function canGen(Lk)
1. Ck+1=Ø
2. for Ii in Lk
3.    for Ij in Lk
4.       if Ii.d1==Ij.d2,…,Ii.dk-1==Ij.dk then
5.       Ck+1=Ck+1∪{{Ij.d1,…,Ij.dk,Ii.dk}}
6. for I in Ck+1
7.    if ∃I′ I′⊆I and length(I′)=k and I′∉Lk then
8.       Ck+1=Ck+1-{I}
9. return Ck+1
```

在恶意序列模式挖掘算法中，C_k 用于存储长度为 k（$k\geq 1$）的候选 API 调用序列模式，L_k 用来存储第 k（$k\geq 1$）轮迭代过程中选出的用于生成 C_{k+1} 的长度为 k 的 API 调用序列模式，MP 用来存储所有挖掘出的 API 恶意调用序列模式。从伪代码中可以看出，算法主要分为 3 个主要部分。第 1 部分（第 5－10 行）扫描样本数据库 DB，对 C_k（$k\geq 1$）中的每个候选序列模式相关的记录进行计数；第 2 部分（第 11－27 行）计算 C_k（$k\geq 1$）中的每个候选序列模式关于恶意类别的支持度 $s\%$ 和置信度 $c\%$，并且根据用户定义的最小支持度 $ms\%$、最小置信度 $mc\%$ 和长度为 $k-1$（$k\geq 2$）的子序列模式的最大置信度的值，选出用于生成 C_{k+1} 的序列模式集 L_k 和恶意序列模式添加到 MP 中；第 3 部分（第 28 行）基于 L_k，通过函数 *canGen* 生成候选序列模式集 C_{k+1}。

需要说明的是，在恶意序列模式挖掘算法的第二部分（第 11－27 行），本文采用 $s\%\geq ms\%$ 和 $c\%>c'\%$ 两个公式进行过滤剪枝，以减少生成的候选 API 调用序列模式的数量。这是因为，在挖掘过程中随着 API 调用序列模式长度的增加，挖掘到的序列模式对恶意样本和正常样本有更强的区分能力，因此，每次迭代过程中都使用公式 $c\%>c'\%$ 来选择对恶意代码有更强辨别能力的 API 调用序列模式，使得 L_{k+1} 比 L_k 中得到的 API 调用序列模式对恶意代码有更强的检测能力。然而，随着 API 调用序列模式长度的增加，挖掘到的序列模式在样本中出现的次数急剧减少，如果不使用公式 $s\%\geq ms\%$ 对序列模式进行剪枝，就会生成大量的出现次数极少的候选序列模式，使得最终得到的分类模型过度拟合样本数据。与此同时，通过使用上述新的过滤剪枝策略，算法在挖掘过程中的运行时间和内存空间消耗得到了显著减少。这也是本文算法相比 GSP 和 OOA 更高效的原因。

本文提出的恶意序列模式挖掘算法基于 GSP 算法挖掘样本中不同类型（连续或者不连续）和长度的 API 子序列模式，保留了样本 API 调用序列的顺序特性；同时结合 OOA 技术来对具有恶意文件类别属性的 API 调用序列模式进行挖掘。相比其他特征形式（迭代模式[11]、API 调用集[9] 和 n-gram模式[13]等），挖掘出的恶意 API 调用序列模式能更好地表示恶意代码的某种潜在恶意行为模式，对恶意 PE 文件和正常 PE 文件具有更好的区分能力。

3.4　恶意代码检测模块

本文使用随机森林算法[18]来建立分类模型并进行恶意代码检测。随机森林算法是一种集成学习算法，在分类实验中表现出比朴素贝叶斯、最近邻和支持向量机等算法更佳的检测效果。

3.4.1　特征表征文件

在构建分类模型对文件进行检测之前，首先需要使用挖掘出的恶意 API 调用序列模式作为特征来表征样本集文件和测试文件，形成布尔向量格式的训练数据集和测试数据集。其中每个布尔向量代表一个特定的文件示例，布尔向量中每个元素的值（0，1）代表此文件是否包含对应的恶意 API 调用序列模式。

3.4.2　检测恶意代码

本文使用随机森林算法建立分类模型并进行恶意代码的检测。随机森林是一种集成学习算法，它通过集成多个决策树分类器的预测给出最终的分类结果。利用随机森林算法得到基础分类器的过程主要分为两步：

（1）对原始训练集进行有放回的取样，直到取得的训练集与原始训练集的大小一致。

（2）假设共有 M 个属性，对于每个内部结点，从所有属性中随机选取 F（$F<M$）个属性作为分裂属性集，并以最佳的分

裂方式对结点进行分裂。

通过原始数据集生成多个决策树分类器后，即可对测试数据进行检测。通过输入测试数据到多个决策树分类器得到预测结果后，随机森林利用简单多数投票法给出最终的检测结果。

4 实验结果和分析

本节通过一系列实验对本文提出的恶意代码检测方法进行评估，并将其与已有的基于 API 调用数据的恶意代码检测方法进行比较。本文所有的实验都在处理器为 Intel Xeon e3-1231 3.4GHz、内存为 8GB 的 Windows 7 旗舰版操作系统上进行。为了获取程序在动态运行过程中调用的真实且完整的 API 调用序列数据，使用处理器为 Intel Xeon e3-1231 3.4 GHz、内存为 8GB、操作系统为 Windows XP SP3 的真机动态执行被分析文件。本文提出的恶意代码检测方法只针对 Windows 平台上 PE 格式的文件进行检测，因为现有的大多数恶意代码都是 PE 文件格式的。

4.1 实验数据集

本文的恶意代码检测方法使用 Windows 平台上的 PE 格式文件作为输入，共收集了 3386 个 Windows PE 格式的样本作为实验数据集，其中包括 2124 个恶意样本和 1262 个正常样本。恶意样本包括逃避型的恶意样本和非逃避型的恶意样本。逃避型的恶意样本共有 1112 个，来自于文献[15]中的实验数据；非逃避型的恶意样本共有 1012 个，下载于开源病毒库 VirusShare[27]。1262 个正常样本则来源于新安装的 Windows XP 系统中的系统文件和不同类型的常用应用程序，如浏览器、文件编辑器、办公软件、媒体播放器等。

4.2 与其他恶意代码检测方法进行比较

本节通过与已有的基于 API 调用数据的恶意代码检测方法进行比较，来评估本文提出的检测方法对未知和逃避型恶意代码的检测能力。为了得到更接近于分类器真实性能的评估结果，使用十折交叉验证来评估不同检测方法对恶意代码的检测效果。本文使用以下评价措施来对实验结果进行比较。

(1)查准率：用于度量检测模型判断为恶意代码的样本中实际为恶意代码的样本所占的比例。查准率越高，检测模型的误报率就越低。

(2)查全率：用于度量恶意代码样本中被检测模型正确判断为恶意代码的样本所占的比例。查全率越高，检测模型的漏报率就越低。

(3)准确率：用于度量全体样本中检测模型判断正确的样本所占的比例。准确率越高，检测模型正确地预测样本的类别的能力就越强。

4.2.1 对未知恶意代码的检测

本实验通过与 Ye 等提出的智能恶意代码检测系统 IMDS[9]进行比较，来评估本文提出的检测方法对未知恶意代码的检测能力。IMDS 已被成功应用于金山杀毒软件中进行恶意代码检测，其具体过程是先对加壳的样本进行脱壳；然后，从每个样本中获取导入函数信息，并使用面向目标的关联挖掘算法来挖掘频繁 API 项集，生成与恶意文件类别相关的关联规则；最后，使用基于关联规则的分类方法来检测恶意代码。本文使用实验数据集中的 1012 个非逃避型恶意样本和 1262 个正常样本来进行对比实验。

本文提出的检测方法 MACSPMD 需要选择与恶意代码相关度高的关键 API 调用进行恶意序列模式的挖掘。如 3.3.1 节所述，通过实验选择最佳的相关性值 r，使得通过条件 $Relevance(a_i, f_M) > r$ 选择出的关键 API 调用集合能表示所有的恶意样本和尽可能少的正常样本。从表 1 的实验结果可知，r 为 0.69 时，所选出的关键 API 调用集合能表示所有的 1012 个恶意样本，同时只能表示 1095 个正常样本，其他的 167 个正常样本的 API 调用序列不包含选出的关键 API。因此，设定最佳的 r 值为 0.69，并对选出的 API 调用进行编号，从而形成关键 API 调用集合。

表 1 不同 r 值选出的关键 API 调用集合对样本的表示能力的比较

Table 1 Representation capabilities comparison of key API call set selected with different r values

r	能表示的恶意样本数的比例/%	能表示的正常样本数的比例/%
0.67	100	88.51
0.68	100	87.48
0.69	100	86.77
0.70	99.80	86.13
0.71	98.91	82.57

本文提出的检测方法和 IMDS 都需要使用 OOA 方法来挖掘恶意模式，并使用挖掘出的恶意模式作为特征进行恶意代码检测。为了确保实验的公正性，统一设置最小支持度 $ms\%$ 为 30%、最小置信度 $mc\%$ 为 93%，来对恶意模式进行充分挖掘，然后选择其中信息增益最大的 1000 个恶意模式作为特征进行恶意代码的检测，实验结果如表 2 所列。

表 2 各检测方法对未知恶意代码的检测效果和恶意模式挖掘所需时间的对比

Table 2 Comparison of detection effects on unknown malware and time required to mine malicious patterns for each detection method

检测方法	恶意模式挖掘所需时间/min	查准率/%	查全率/%	准确率/%
MACSPMD	7.4	90.73	97.73	94.55
IMDS	3312	87.89	95.35	92.08

表 2 中的实验结果表明，本文提出的恶意代码检测方法 MACSPMD 相对于 IMDS 检测方法对未知恶意代码的检测效果更佳。这是因为，IMDS 使用面向目标的关联挖掘算法生成无序的恶意 API 调用频繁项集来进行恶意代码检测，而本文结合序列挖掘技术和面向目标的关联挖掘方法来对恶意 API 调用序列模式进行挖掘，保留了 API 调用具有的顺序特性。此外，从表 2 中恶意模式挖掘所需时间的结果可以看出，本文提出的恶意代码检测方法在特征挖掘过程中消耗的时间远远少于 IMDS。这主要有两方面的原因：1)IMDS 直接使用从样本文件中获取的全部 API 来进行恶意频繁 API 项集的挖掘，而本文先从全部 API 调用集合中选择与恶意代码相关度高的关键 API，再使用选出的关键 API 调用集合来进行恶意序列模式的挖掘，减少了用于恶意模式挖掘的初始 API 调用的数量，从而减少了挖掘过程的时间消耗；2)本文在挖掘过

程中除了采用 IMDS 中的 $s\%\geqslant ms\%$（支持度）和 $c\%\geqslant mc\%$（置信度）两个剪枝条件外，还使用公式 $c\%>c'\%$（置信度增量）对形成的冗余序列模式分支进行剪枝，剪掉在挖掘过程中随着 API 调用序列模式长度的增加对恶意样本和正常样本的区分能力没有增强的序列模式分支，从而避免在挖掘过程中产生大量冗余且无益的 API 调用序列模式，极大地减少了挖掘过程消耗的时间。总体来说，本文提出的恶意代码检测方法能有效地检测未知恶意代码。

4.2.2　对逃避型恶意代码的检测实验

本实验中，通过与 Naval 等[16]提出的逃避型恶意代码检测方法进行对比，来评估本文提出的检测方法对逃避型恶意代码进行检测的能力。文献[16]中提出的逃避型恶意代码检测方法的具体过程如下：首先，收集文件在硬件虚拟化平台 Ether[17]中动态执行所产生的系统调用序列；然后，应用马尔科夫链计算相邻系统调用间转换的概率，并把每个文件转换成一个 284×284（Windows XP SP3 系统中共有 284 个系统调用函数）的转换概率矩阵（Transition Probability Matrix，TPM）；接着，将所有样本的 TPM 相加形成一个合成矩阵，并从中选取取值最大的 m 个转换状态，即从相邻系统调用组成的 2-gram 调用序列模式中选择出现频率最大的 m 个 2-gram 相邻系统调用序列模式作为特征；最后，从每个 TPM 中获取选出的 m 个 2-gram 相邻系统调用序列对应的转换概率来形成输入向量，同时使用神经网络算法建立分类模型来检测恶意代码并判断具体的逃避类型。本文使用实验数据集中的 1112 个逃避型恶意样本和 1262 个正常样本来进行对比实验。

对于本文所提检测方法 MACSPMD，在实验过程中使用 3.3.1 节的方法来设定最小相关性值 r，从而选出关键 API 调用集。从表 3 的实验结果可知，r 为 0.86 时，选出的关键 API 调用集合能表示所有的 1112 个逃避型恶意样本，同时只能表示 1197 个正常样本，其他的 65 个正常样本的 API 调用序列不包含选出的关键 API。因此，设定最佳的 r 值为 0.86。

表 3　不同 r 值选出的关键 API 调用集合对样本的表示能力比较

Table 3　Comparison of representation capabilities of key API call set selected with different r values

r	能表示的恶意样本数的比例/%	能表示的正常样本数的比例/%
0.84	100	96.75
0.85	100	95.80
0.86	100	94.85
0.87	99.73	94.69
0.88	97.57	92.07

在本文检测方法 MACSPMD 的恶意 API 调用序列模式挖掘过程中，设置最小支持度 $ms\%$ 为 0%、最小置信度 $mc\%$ 为 95%来对恶意序列模式进行充分挖掘。为了充分对比不同长度的恶意序列模式作为特征进行逃避型恶意代码检测的效果，分别从所有挖掘出的恶意序列模式中选择信息增益最大的 200 个恶意序列模式和从长度为 $l(1\leqslant l\leqslant 8)$ 的恶意序列模式中选择信息增益最大的 m 个（如果挖掘出的长度为 l 的恶意序列模式的个数 $n\geqslant 200$，则 $m=200$，否则 $m=n$）恶意序列模式作为特征进行逃避型恶意代码的检测。为了保证测试的公平性，在对 Naval 等提出的检测方法进行实验的过程中，取 $m=200$，即选取合成矩阵中取值最大的 200 个 2-gram 相邻系统调用序列模式（序列长度 $l=2$）作为特征，使用其对应的转换概率形成输入向量，进行逃避型恶意代码的检测。实验结果如表 4 所列。

表 4　各检测方法对逃避型恶意代码的检测效果的对比

Table 4　Comparison of detection effects on evasive-malware for each detection method

实验编号	检测方法(l,m)	查准率/%	查全率/%	准确率/%
1	MACSPMD(1～8,200)	96.32	98.92	97.73
2	MACSPMD(1,154)	86.83	90.11	88.96
3	MACSPMD(2,200)	92.59	97.75	95.28
4	MACSPMD(3,200)	93.73	98.11	96.04
5	MACSPMD(4,200)	94.00	98.65	96.42
6	MACSPMD(5,135)	91.84	96.13	94.19
7	MACSPMD(6,31)	87.29	89.57	89.00
8	MACSPMD(7,6)	78.92	81.12	81.00
9	MACSPMD(8,2)	70.85	72.12	73.04
10	Naval 等所提方法(2,200)	92.49	97.39	95.07

表 4 中的实验 1 和实验 10 的结果表明，相对于 Naval 等人提出的检测方法，本文提出的恶意代码检测方法对逃避型恶意代码的检测效果更佳。这是因为，Naval 等人仅使用相邻系统调用这种长度为 2 的相邻序列模式作为特征进行恶意代码的检测，虽然这种方法已经取得了不错的检测效果，但本文结合序列挖掘技术和面向目标关联挖掘方法挖掘出的恶意 API 调用序列模式不仅包含了相邻系统调用这种序列模式，还包含了不连续类型和其他长度的序列模式。图 3 为实验 1 中选出的 200 个序列模式的长度分布图，其中长度为 2～5 的序列模式占比较大。由表 4 中实验 3 和实验 10 的结果可知，在特征个数相同、序列模式长度也相同的情况下，由于 MACSPMD 挖掘到的序列模式包含连续和不连续两种子序列特征，因此其能更好地表示恶意代码潜在的恶意行为模式。对比表 4 中实验 2－实验 9 的结果可知，在选择的特征个数相同的情况下，使用本文 MACSPMD 方法挖掘到的长序列模式对恶意样本和正常样本往往具有更强的区分能力。此外，Naval 等人提出的检测方法使用 Ether 作为动态分析环境，对于某些能够检测到 Ether 分析框架的逃避型恶意样本，本文使用的真机分析环境往往能在样本的动态运行过程中获取到更真实而完整的 API 调用序列数据，达到更好地检测逃避型恶意代码的目的。综上，本文提出的恶意代码检测方法能有效地检测包括逃避型在内的恶意代码。

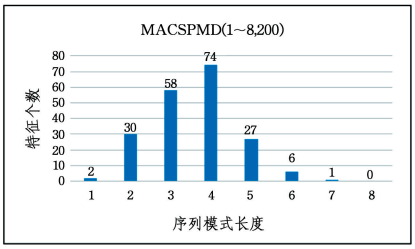

图 3　MACSPMD(1～8,200)中序列模式的长度分布图

Fig. 3　Length distribution diagram of sequence patterns in MACSPMD(1～8,200)

结束语　本文对现有基于动态 API 调用的恶意代码检测方法和相关工作进行了分析和回顾，并针对当前检测方法存在的缺陷和不足提出了一种新的恶意代码检测方法，它能

自动化检测包括逃避型在内的已知和未知恶意代码。本文提出的恶意代码检测方法首先使用真机分析环境获取文件动态运行过程中调用的真实而完整的 API 调用序列；然后，选取与恶意代码相关度高的 API 调用形成关键 API 调用集，并使用本文提出的恶意序列模式挖掘算法挖掘出恶意 API 调用序列模式；接着，使用挖掘出的恶意 API 调用序列模式来表征样本文件，并使用随机森林算法来构建恶意代码的检测模型；最后，使用挖掘出的恶意 API 调用序列模式来表征测试文件，并使用恶意代码检测模型进行恶意代码的检测。在真实数据集上的实验结果表明，本文提出的恶意代码检测方法在检测未知和逃避型恶意代码方面优于其他基于 API 调用数据的恶意代码检测方法。

本文提出的恶意代码检测方法 MACSPMD 没有考虑 PE 文件的多路径执行问题，因此不能有效检测满足一定触发条件才执行恶意行为的基于触发器的恶意软件。在以后的工作中，可以对文件的多路径执行进行研究，在分析环境中加入相应的多路径执行和监控模块来对基于条件触发的恶意软件进行检测。此外，本文只对 MACSPMD 方法在恶意代码的检测方面进行了实验研究与应用，并未在恶意代码分类问题上进行扩展和应用。但是，理论上 MACSPMD 可以对不同类别的恶意代码的恶意 API 调用序列模式进行挖掘，并将这些模式作为特征进行恶意代码的分类。在以后的工作中，我们将对本文提出的 MACSPMD 进行扩展，进行恶意代码分类的相关研究。

参 考 文 献

[1] PHILIP O, SEZER S, MCLAUGHLIN K. Obfuscation: The Hidden Malware[J]. IEEE Security & Privacy, 2011, 9(5): 41-47.

[2] ROUNDY K A, MILLER B P. Binary-code obfuscations in prevalent packer tools[J]. Acm Computing Surveys, 2013, 46(1): 1-32.

[3] MURAD K, SHIRAZI N U H, ZIKRIA Y B, et al. Evading Virus Detection Using Code Obfuscation[M]// Future Generation Information Technology. Springer Berlin Heidelberg, 2010: 394-401.

[4] WU D F, WANG C G, HAO X W. Study on metamorphic technique of malware[J]. Computer Applications and Software, 2012, 29(3): 74-77. (in Chinese)
吴丹飞，王春刚，郝兴伟. 恶意代码的变形技术研究[J]. 计算机应用与软件，2012，29(3)：74-77.

[5] EGELE M, SCHOLTE T, KIRDA E, et al. A survey on automated dynamic malware-analysis techniques and tools[J]. ACM Computing Surveys, 2008, 44(2): 1-42.

[6] SRIKANT R, AFRAWAL R. Mining sequential patterns: Generalizations and performance improvements[C]// International Conference on Extending Database Technology. Springer Berlin Heidelberg, 1996: 1-17.

[7] SHEN Y D, ZHANG Z, YANG Q. Objective-Oriented Utility-Based Association Mining[C]// IEEE International Conference on Data Mining, 2002. IEEE, 2002: 426-433.

[8] MASUD M M, KHAN L, THURAISINGHAM B. A scalable multi-level feature extraction technique to detect malicious executables[J]. Information Systems Frontiers, 2008, 10(1): 33-45.

[9] YE Y, WANG D, LI T, et al. An intelligent PE-malware detection system based on association mining[J]. Journal in Computer Virology, 2008, 4(4): 323-334.

[10] FAN Y, YE Y, CHEN L. Malicious sequential pattern mining for automatic malware detection[J]. Expert Systems with Applications, 2016, 52(C): 16-25.

[11] AHMADI M, SAMI A, RAHIMI H, et al. Malware detection by behavioural sequential patterns[J]. Computer Fraud & Security, 2013, 2013(8): 11-19.

[12] HAN L S, GAO K L, ZHAO B H, et al. Behavior detection of malware based on combination of API function and its parameters[J]. Application Research of Computers, 2013, 30(11): 3407-3410. (in Chinese)
韩兰胜，高昆仑，赵保华，等. 基于 API 函数及其参数相结合的恶意软件行为检测[J]. 计算机应用研究，2013，30(11)：3407-3410.

[13] LI M, JIA X Q, WANG R, et al. A feature selection and modelling method for malicious code[J]. Computer Applications and Software, 2015, 32(8): 266-271. (in Chinese)
李盟，贾晓启，王蕊，等. 一种恶意代码特征选取和建模方法[J]. 计算机应用与软件，2015，32(8)：266-271.

[14] BALZAROTTI D, COVA M, KARLBERGER C, et al. Efficient Detection of Split Personalities in Malware[C]// Network and Distributed System Security Symposium (NDSS 2010). San Diego, California, USA, DBLP, 2010.

[15] KIRAT D, VIGNA G, KRUEGEL C. BareCloud: Bare-metal Analysis-based Evasive Malware Detection[C]// USENIX Security Symposium. 2014: 287-301.

[16] NAVAL S, LAXMI V, GAUR M S, et al. Environment-Reactive Malware Behavior: Detection and Categorization[M]// Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance. Springer International Publishing, 2015: 167-182.

[17] DINABURG A, ROYAL P, SHARIF M, et al. Ether: malware analysis via hardware virtualization extensions[C]// ACM Conference on Computer and Communications Security (CCS 2008). Alexandria, Virginia, USA, DBLP, 2008: 51-62.

[18] BREIMAN L. Random forests[J]. Machine Learning, 2001, 45(1): 5-32.

[19] Cuckoo Sandbox[EB/OL]. [2017-06-17]. http://www.cuckoosandbox.org.

[20] Pafish[EB/OL]. [2017-06-17]. https://github.com/a0rtega/pafish.

[21] Anubis[EB/OL]. [2017-06-17]. http://anubis.iseclab.org.

[22] WILLEMS C, HOLZ T, FREILING F. Toward Automated Dynamic Malware Analysis Using CWSandbox[J]. IEEE Security & Privacy, 2007, 5(2): 32-39.

[23] Clonezilla[EB/OL]. [2017-06-17]. http://www.clonezilla.org.

[24] Acronis True Image[EB/OL]. [2017-06-17]. http://www.acronis.com.

[25] NortonGhost[EB/OL]. [2017-06-17]. http://www.symantec-norton.com/Norton_Ghost_15.0_p115.aspx.

[26] INetSim[EB/OL]. [2017-06-17]. http://www.inetsim.org.

[27] VirusShare[EB/OL]. [2017-06-17]. https://virusshare.com.