基于高斯混合模型的增量聚类方法识别恶意软件家族

胡建伟[1]，车欣[1]，周漫[2]，崔艳鹏[1]

（1. 西安电子科技大学网络与信息安全学院，陕西 西安 710071；2. 华中科技大学网络空间安全学院，湖北 武汉 430074）

摘 要：针对属于同一个家族的恶意软件的行为特征具有逻辑相似性这一特点，从行为检测的角度通过追踪 API 函数调用的逻辑规则来提取恶意软件的特征，并利用静态分析与动态分析相结合的方法来分析恶意行为特征。此外，依据恶意软件家族的目的性、继承性与多样性，构建了恶意软件家族的传递闭包关系，并改进了基于高斯混合模型的增量聚类方法来识别恶意软件家族。实验证明，所提方法不仅能节省恶意软件检测的存储空间，还能显著提高检测的准确率与识别率。

关键词：恶意软件家族；高斯混合模型；增量聚类；API 函数调用；逻辑规则

中图分类号：TP393

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2019135

Incremental clustering method based on Gaussian mixture model to identify malware family

HU Jianwei[1], CHE Xin[1], ZHOU Man[2], CUI Yanpeng[1]

1. School of Network and Information Security, Xidian University, Xi'an 710071, China

2. Institute of Cyberspace Security, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract: Aiming at the logical similarity of the behavioral characteristics of malware belonging to the same family, the characteristics of malware were extracted by tracking the logic rules of API function call from the perspective of behavior detection, and the static analysis and dynamic analysis methods were combined to analyze malicious behavior characteristics. In addition, according to the purpose, inheritance and diversity of the malware family, the transitive closure relationship of the malware family was constructed, and then the incremental clustering method based on Gaussian mixture model was improved to identify the malware family. Experiments show that the proposed method can not only save the storage space of malware detection, but also significantly improve the detection accuracy and recognition efficiency.

Key words: malware family, Gaussian mixture model, incremental clustering, API function call, logic rule

1 引言

近年来，利用恶意软件进行网络攻击的行为越来越多[1]。恶意软件利用欺骗技术可以在发动攻击的同时逃避反病毒检测，具有多态性、隐蔽性、易感染性等特性，严重影响网络数据或程序的安全性、使用性与整合性，给互联网和用户带来巨大威胁，造成严重的损失，因此，恶意软件检测技术已经成为目前信息安全人员的研究热点之一。

然而，当前的恶意软件检测技术存在高误报率和高漏报率的不足[2]，难以检测出采用了欺骗技术的恶意软件。值得注意的是，目前流行的恶意软件都具有很强的目的性，恶意代码编写者依据已有的恶意软件不断开发出行为目的相似但代码结构又不完全相同的恶意软件，从而形成恶意软件家族。据研究结果证实[3]，超过 98%的新恶意软件样本实

收稿日期：2018−12−07；**修回日期**：2019−05−21

通信作者：周漫，zhou_man1125@hust.edu.cn

基金项目：国家自然科学基金资助项目（No.61272033）

Foundation Item: The National Natural Science Foundation of China (No.61272033)

际上是来自现有恶意软件系列的“衍生物”，新的恶意软件继承了原始恶意软件的部分功能。为了躲避检测并快速地部署恶意软件，黑客通常不会重新开发新的恶意软件，而是改进恶意软件现有的行为逻辑或者在现有的恶意软件中添加新的恶意行为逻辑，即新的恶意软件具有继承性与多态性。本文将具有相似行为逻辑或者相同行为目的的恶意软件集合称为恶意软件家族。

为了提高恶意软件检测的准确率与检测效率，本文提出了基于高斯混合模型（GMM, Gaussian mixture model）的增量聚类方法来识别恶意软件家族。本文的主要工作如下。

1) 依据属于同一个家族的恶意软件的行为特征具有逻辑相似性这一特点，本文从行为检测的角度分析并识别恶意软件家族。

2) 为了构建恶意行为特征的分析框架，本文利用静态分析与动态分析相结合的方法来提取 API 函数调用的抽象特征，通过分析 API 函数调用的参数依赖关系来构建恶意软件行为逻辑图。

3) 为了找到拥有整个软件家族恶意行为特征的恶意软件群 U_M 与拥有软件家族成员共有的恶意行为特征的恶意软件群 C_M，本文依据恶意软件家族行为的继承性与多样性，为特定目的的恶意软件家族构建 4 个行为传递闭包，并建立特征行为与恶意软件的一对一映射关系。

4) 针对传统聚类方法不能利用上一次聚类结果，从而导致耗时、资源浪费等问题，本文采用基于高斯混合模型的增量聚类方法来识别恶意软件家族，创建并动态调整与恶意软件家族的进化史相一致的高斯混合模型树，并引入增量学习，同时进行恶意软件家族的识别与恶意样本的聚类。

2　研究背景和相关工作

随着当前恶意软件的欺骗技术越来越成熟，以及各类病毒数量的急剧增加，导致传统的恶意软件检测技术不再有效。因此，出现了各种基于行为的恶意软件检测技术。Pektas 等[4]通过 API 调用序列挖掘和搜索 n-gram 从而收集代表恶意软件行为特征的集合。针对目前恶意软件识别率下降的现状，Han 等[5]指出造成这种困境的原因是越来越多的目的性恶意软件攻击已经出现，与传统恶意软件几乎没有共同特征。Han 等基于可判定理论，证明了任何软件执行的任务都是递归的和可确定的，并通过建立从软件到任务的多对一的映射，证明了包括恶意软件在内的各类软件也是递归的，并且可以由相应的任务来确定。

为了提高检测恶意软件的准确率，Kolosnjaji 等[6]提出首先在沙箱中执行恶意软件样本以收集系统调用，然后使用深度神经网络对恶意软件的系统调用序列进行建模以用于恶意软件分类。Cho 等[7]利用动态行为分析工具将 API 序列提取为恶意软件行为报告，然后使用 Malheur 进行聚类和分类分析。

近年来，基于机器学习和数据挖掘算法的恶意软件行为特征的分析方法逐渐受到研究人员的重视。Santos 等[8]提出使用可执行文件的操作码序列频率来检测和分类恶意软件，通过这种方式来训练机器学习算法从而检测未知的恶意软件变种。Arp 等[9]将针对 API 函数的静态分析与机器学习算法相结合，以检测恶意软件。他们在向量空间中嵌入了特征，从向量空间中发现了恶意软件模型，并使用这些模型构建了机器学习检测系统。

传统的聚类方法主要是利用批处理模型来发现固定特征数据库的数据集群，但是目前出现了越来越多的动态数据集，数据点以流形方式输入。在这种情况下，增量聚类可以有效地处理这样的数据集[10]。当不断输入数据点时，增量聚类逐步更新聚类结果，使当前的所有数据存在一个最新的聚类。为了对流数据进行数据聚类，Wan 等[11]提出了一种基于高斯混合模型的新型增量聚类方法，称为 ICGT（incremental clustering of GMM tree）。ICGT 创建并动态调整与数据流顺序一致的 GMM 树，树中的每个叶子节点对应于密集高斯分布，每个非叶子节点对应于 GMM。为了更新 GMM 树以插入新输入的数据点，Wan 等引入了节点连接和连接子集的定义，并提出了树更新算法，实验结果证实所提方法是有效的。

3　恶意软件家族识别

基于软件家族恶意行为的依赖性与继承性[12]，人们能为每个恶意软件家族建立一个特征库，并挑选出具有代表性的恶意软件集合。当出现未知的恶意软件时，人们可以提取它的特征，并与最具代表性的恶意软件集合的特征进行比对，如果具有该家族的恶意签名或特征，则此未知的恶意软件属于该恶意软件家族；否则，需要分析软件的行为特征，将分析出来的有意义的特征加入特征库中再进行

聚类[13]。本文将新样本划分到一个已知的恶意软件家族的过程定义为聚类过程，并将依据新样本的行为特征更新已知的恶意软件特征库，直到最后发现新的家族并更新已知家族成员的过程定义为恶意软件家族识别。同时执行恶意软件识别与恶意聚类，提高恶意软件的检测效率。

3.1 恶意行为的继承性与逻辑性

若追根溯源，每一个恶意软件家族一定会有“祖先”[3]。随着时间推移，恶意代码需要不断“衍生”，不断更新技术以满足功能需求，逃避恶意软件检测，但“衍生”的恶意软件与其祖先的行为特征相似，这体现了恶意软件家族的继承性。为了躲避检测并快速地部署恶意软件，恶意代码创建者通常不会重新开发新的恶意软件，而是在已有的恶意代码的基础上运用欺骗技术，改进恶意软件现有的行为逻辑或者添加新的恶意行为逻辑。欺骗技术包括静态欺骗技术与动态欺骗技术两方面[14]。静态欺骗技术包括：1) 代码混淆，一种在不改变程序功能的前提下，将正常源代码转换成更难以阅读和理解的形式的技术；2) 加壳，一种为了保护目标程序，而在运行时先于目标程序运行的一段代码，常见的压缩壳有 UPX、Aspack 等，常见的加密壳有 ASProtect、Armadillo 等；3) 自修改代码（SMC, self modifying code），一种对程序核心代码和数据进行加密，且在被加密代码执行前，才进行解密的一种技术。动态欺骗技术包括：1) 反调试技术，当恶意代码检测到程序正在一个调试器上运行时，它会修改自身代码来躲避调试；2) 反虚拟机技术，恶意代码在运行前会检测自己是否运行在虚拟机中，如果检测到正在虚拟机中运行，恶意代码会执行与其本身行为不同的行为，其中最简单的行为是停止运行。

巴西恶意代码进化史大致可分为 3 个阶段：1) 初级阶段，在这个阶段，恶意代码就是一段开源的键盘记录器代码，没有使用任何反分析机制；2)中级阶段，恶意代码开发者开发出鼠标记录器和钓鱼木马，并且为了钓鱼木马不轻易被识别出来，开发者修改 host 文件，将银行网站的域名解析到一个硬编码的服务器上；3)高级阶段，恶意代码开发者使用 Internet explorer 自动化来操作转账页面内容，同时为了延长恶意代码的生存周期，恶意代码开发者采用代码混淆、反调试技术、反虚拟机技术等来躲避反病毒软件的检测。每个阶段的代码实现方式及采用的恶意欺骗手段如表 1 所示。

表 1 巴西恶意代码进化史

阶段	恶意代码形式	实现方式	反分析机制
初级	键盘记录器	GetKeyState GetAsyncKeyState	无
中级	鼠标记录器和钓鱼木马	DdeQueryString LocalAlloc DdeQueryString	修改 host 文件
高级	浏览器帮助对象	SetSite RetrieveBrowserWindow FindConnectionPoint	代码混淆、反调试技术、反虚拟机技术

当前的恶意软件有很强的目的性，它们通过已有的恶意代码不断开发出行为目的相似但代码结构不完全相同的恶意软件，从而形成恶意软件家族。依据恶意软件行为的目的性，本文从行为检测的角度分析并识别恶意软件家族，将恶意软件家族分为 8 类，分类框架如表 2 所示。该检测方法的优势在于不管恶意软件的结构如何复杂、数量如何庞大，其根本的行为特征都具有相似的逻辑性，从而使基于行为的检测方法可以有效地对已知和未知的恶意代码进行鉴别和检测。这种方法一方面可以提高检测效率和检测成本，另一方面可以避免传统的病毒检测技术，只有等到计算机被感染时才能实现检测的弊端，实现病毒快速、准确的防御。

表 2 基于恶意目的的恶意软件分类框架

分类	目的
间谍程序	未经用户允许的情况下控制系统资源
蠕虫	可主动自我传播和自我执行
木马	盗取被感染主机的敏感信息
后门	对受害者的计算机进行远程秘密访问
Rootkit	自我复制并感染其他程序
病毒	在用户不知情的情况下控制系统资源
勒索	加密和劫持用户主机的关键文件以骗取赎金
恶意挖矿	未经授权使用别人的计算机挖掘“加密货币”

3.2 恶意软件家族的行为特征

基于恶意软件行为的检测方法分为基于行为的精确匹配和模糊匹配，其中精确匹配方法对目前的恶意软件几乎不起作用，因为当前的恶意代码大多利用恶意欺骗技术来伪装自己以逃避防火墙和反病毒软件的检测，它们会经过一层层封装后再执行，因此模糊匹配成为主要的判别方法，其中利用 API 函数调用来识别恶意行为是比较常用的方法[15]。

恶意程序会以异常的频率调用实现特定功能的或者罕见的 API 函数序列，或者在正常软件中不常使用的 API 函数。因此，分析样本软件在运行时 API 的调用情况可以有效地鉴别样本软件。

提取 API 调用序列可以通过 2 种途径，静态分析与动态分析。静态分析是指不运行可执行文件，而是通过反汇编目标文件来提取有用的低级行为特征，如字节序列、操作码等。高级行为特征可以通过分析 API 调用序列、函数功能、控制流程图来提取。但是静态分析会受到代码混淆、加密和加壳等欺骗技术的影响。动态分析是指在恶意代码运行时提取行为特征，提供更有效的结果。通过在虚拟机中运行可执行文件样本，跟踪程序的执行过程并分析其恶意性，能够有效地检测出使用欺骗技术的恶意软件。此外，还可以利用沙箱技术在线分析病毒，检测软件恶意行为，已知的沙箱 Cuckoo 可以有效地分析和处理未知的恶意软件[16]。但是动态分析仅着眼于恶意代码在当前实验环境中的行为，忽略了恶意代码程序本身的代码，因此会受实验环境的限制，进而有可能无法得出样本真实的恶意行为特征。因此，本文采用静态分析与动态调试联合分析的方法，具体如图 1 所示。

本文将静态分析与动态分析相结合来分析恶意软件并提取其特征。在静态分析部分，本文首先对样本进行预处理，然后利用 Python 的 pefile 模块来提取样本导入的 API。具体操作是：首先，解析样本 PE 文件；然后，通过属性 DIRECTORY_ENTRY_IMPORT 遍历样本文件所有导入的 dll；最后，通过属性 imports 遍历所有的导入函数。但是单纯通过分析样本文件导入表中的 API 函数来判断样本的行为是不准确的，因为恶意软件的开发者可通过函数 LoadLibrary 和 GetProcAddress 来动态加载所需要的 API 函数，所以还需继续进行动态分析。

在动态分析部分，本文利用一个动态 API 函数调用跟踪器 Drltrace 来提取样本的动态 API 调用序列、参数和返回值。Drltrace 是基于 DBI[17]的、主要用于 Windows 和 Linux 的恶意代码分析，其优点是运行、分析恶意代码足够快，能够有效对抗基于时间的反调试技术，并且支持所有类型的库连接，能有效对抗多种反分析机制。获取 API 调用序列后，本文使用污点传播分析技术来提取样本的动态行为特征。首先标记污染源，然后根据 API 函数调用流程，跟踪被标记为污点的数据在整个程序中的传播路径。使用污点传播分析技术，可以清晰地观察到污点在整个程序中的传播路径，其本质是可以反映参数变量在程序中的传播过程。

以 3 个阶段的巴西恶意代码为例，本文首先利用静态分析得到部分的 API 函数，然后再结合动态分析得到恶意代码完整的 API 调用序列、参数及返回值，最后得到 3 个阶段的巴西恶意代码的部分 API 调用序列及其参数依赖性，如图 2 所示。图 2 中加粗显示的函数是实现窃取用户信息的主要功能函数，相同的参数是通过追踪器 Drltrace 与污点传播技术分析后得到的。

同一恶意软件家族具有相似的行为逻辑，而恶意软件 API 函数的调用也具有强逻辑性。表 3 展示了恶意代码家族 Wannacry、NotPetya 和 Kuzzle 的 API 调用逻辑规则。结合图 2 的分析可以得到，API 调用存在参数依赖性，函数调用在时序上有特定的

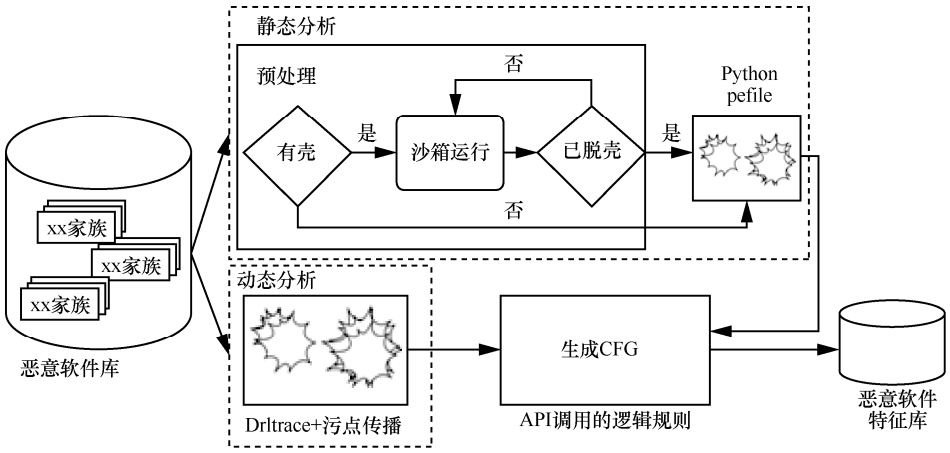


图 1　恶意软件分析框架

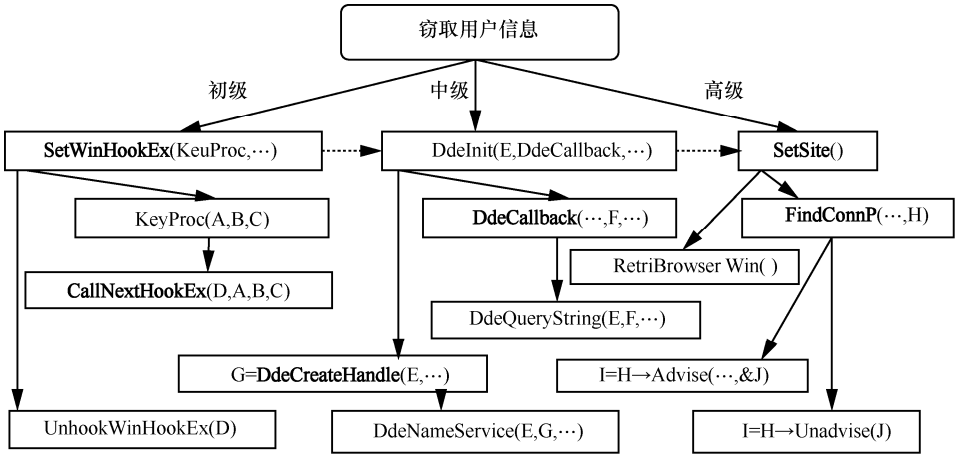


图 2　API 调用序列及参数

逻辑，因此，本文通过分析恶意软件 API 调用的逻辑规则来提取行为特征。

表 3　恶意代码家族的 API 调用逻辑规则

恶意代码家族	行为特征	API 调用逻辑规则
Wannacy	读取资源	$F=f_1+f_2+f_3+f_4$ f_1: hRsrc=FindResource(…); f_2: hGlobal=LoadResource(…,hRsrc); f_3: hGlobal=LoadResource(hGlobal); f_4: SizeofResource(…,hRsrc) ;
NotPetya	进程提权	$F=f_1+f_2+f_3+f_4$ f_1: OpenProcessToken(…, & hToken); f_2:LookupPrivilegeValue(…); f_3: AdjustTokenPrivileges(hToken, …); f_4: CloseHandle(hToken);
Kuzzle	读取网页源码	$F=f_1+f_2+f_3+f_4$ f_1:hInt1=InttOpen(…); f_2: hInt2=OpenUrl(hInt1, …); f_3: IntReadFile(hInt2, …); f_4: CloseHandle(hInt1,hInt2);

3.3　恶意软件行为特征的量化

本文从行为检测的角度分析并识别恶意软件家族，依据恶意软件家族的目的将其分为 8 类，任意软件均可以依据其行为目的、技术手段与危害性得到量化后的行为特征轨迹。本文构建一个四维空间来表示恶意软件所使用的不同恶意技术，即将软件行为（获取、创建、跟踪、破坏）量化为一个四元组 $b=(f(a),f(c),f(t),f(d))$。并依据其目的，将整个空间划分为 8 个小空间，不同目的的恶意行为属于不同的小空间。为了简化计算，本文设置具有不同目的的软件（如间谍软件、蠕虫、木马、后门、Rootkit、病毒、勒索软件、恶意挖矿软件等）行为特征，其四元组分量的取值范围分别为[0,1)、[1,2)、[2,3)、[3,4)、[4,5)、[5,6)、[6,7)、[7,8)，即不同目的的软件的特征节点位于不同的空间。空间中的节点是软件的特征节点，代表软件的行为，特征节点的位置取决于软件行为量化后四元组的值。软件行为的危害性越大，则软件行为特征的四元组取值越大，表现为特征节点的半径越大。集合软件所有的特征节点能得到软件的行为轨迹。

图 3 为间谍软件类部分恶意软件经量化后得到的行为特征，坐标轴为量化的四元组。间谍软件类软件行为特征的四元组分量取值为[0,1)，取值越大，表示其危害性越大。图 3 中的 3 条曲线分别表示表 1 中巴西恶意软件 3 个阶段的行为特征轨迹。图 3 中初级阶段的恶意软件行为特征可用 4 个四元组来表示：{[0.12,0.08, 0.16, 0.09]，[0.24,0.12,0.09,0.13]，[0.13,0.17, 0.23, 0.36]，[0.15,0.32,0.31,0.36]}。

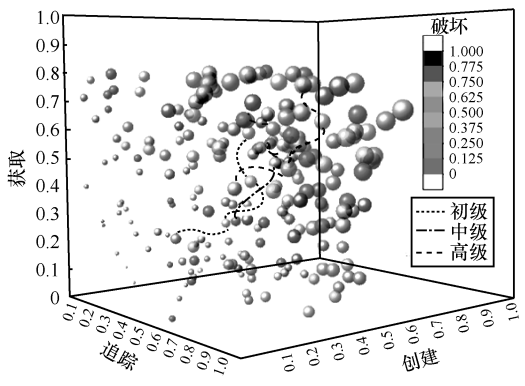


图 3　间谍软件特征的量化空间

为了量化软件行为特征，本文利用静态分析与动态分析相结合的方法来分析恶意软件行为，

然后提取软件的 L 个行为，构成 L 个四元组。软件的特征轨迹一定属于 8 个小空间中的一个，体现了恶意软件家族的目的性；在同一个家族中具有“衍生”关系的软件的行为特征轨迹会有交集，揭示了恶意软件家族的继承性；继承了“祖先”部分代码并“衍生”出新的恶意技术的后代软件具有更严重的危害性，其特征节点半径更大，体现了恶意软件的多样性。

3.4　构建恶意软件家族的传递闭包关系

为了能够有效地度量具有继承性与多样性的恶意软件的相似性，本文将恶意软件的“继承”与“衍生”过程定义为一种传递闭包关系，并找到能代表所属的恶意软件家族的恶意软件集合。对于任意关系 R，R 的传递闭包总是存在的，具有传递关系的任意家族的交集也是传递的。具体的恶意软件家族的传递闭包关系定义过程如下。

恶意软件的行为（获取、创建、跟踪、破坏）用四元组 $b=(f(a),f(c),f(t),f(d))$来表示。若恶意软件家族初始的“族长”R_0 的恶意行为特征为$(f(a_0),f(c_0),f(t_0),f(d_0))$，且由 R_0 派生的恶意软件为 R_{11} 与 R_{12}，其恶意行为特征为$(f(a_{101}),f(c_{101}),f(t_{101}),f(d_{101}))$与$(f(a_{102}),f(c_{102}),f(t_{102}),f(d_{102}))$。则$(f(a_0),f(a_{101}))$、$(f(c_0),f(c_{101}))$、$(f(t_0),f(t_{101}))$、$(f(d_0),f(d_{101}))$、$(f(a_0),f(a_{102}))$、$(f(c_0),f(c_{102}))$、$(f(t_0),f(t_{102}))$与$(f(d_0),f(d_{102}))$为传递关系，且针对四元组 $b=(f(a),f(c),f(t),f(d))$，具有特定目的的恶意软件家族的行为特征可以构成 4 个传递闭包关系：$T(f(a_{ij}),f(a_{i+1jm}))$、$T(f(c_{ik}),f(c_{i+1kn}))$、$T(f(t_{iy}),f(t_{i+1yp}))$与 $T(f(d_{iz}),f(d_{i+1zq}))$。若“衍生”$N$ 代，则 $i\in(0,N)$。m、n、p、q 分别是行为特征为 $f(a_{ij})$、$f(c_{ik})$、$f(t_{iy})$、$f(d_{iz})$的样本的第 m、n、p、q 个衍生对象，j、k、y、z 分别是恶意行为特征$f(a_{i+1jm})$、$f(c_{i+1kn})$、$f(t_{i+1yp})$、$f(d_{i+1zq})$的派生史。

$$T_1=\bigcup_{e,h,r,s,w,x,g,l}(f(a_{New}),f(c_{Nhx}),f(t_{Nra}),f(d_{Nsl}))$$
$$T_2=\bigcap_{e,h,r,s,w,x,g,l}(f(a_{New}),f(c_{Nhx}),f(t_{Nra}),f(d_{Nsl})) \quad (1)$$

其中，T_1 表示某个恶意家族的恶意行为特征的集合，T_2 表示该恶意家族成员共有的恶意行为特征的集合，并且$(f(a_{New}),f(c_{Nhx}),f(t_{Nrg}),f(d_{Nsl}))$属于$(f(a),f(c),f(t),f(d))$的第 N 代衍生对象中的任意一个。由 Han 等[5]的研究可知，$\forall e,h,r,s,w,x,g,l$，必有一个或多个恶意软件与恶意行为特征$(f(a_{New}),f(c_{Nhx}),f(t_{Nrg}),f(d_{Nsl}))$相对应，即可以建立行为与恶意软件的一对多的映射关系。在 T_1 中，对 $\forall e,h,r,s,w,x,g,l$，在$(f(a_{New}),f(c_{Nhx}),f(t_{Nrg}),f(d_{Nsl}))$所对应的多个恶意软件中任意选取一个，并建立一一映射关系，如式(2)所示。

$$(f(a_{n_te_iw_i}),f(c_{n_ih_ix_i}),f(t_{n_ir_ig_i}),f(d_{n_is_il_i}))\Leftrightarrow M_i \quad (2)$$

然后，由 T_1 中所有的恶意行为特征所对应的恶意软件来构成集合 U_M，则集合 U_M 可以代表整个恶意软件家族，如式(3)所示。

$$U_M\Leftrightarrow\sum_i M_i \quad (3)$$

所以新的不定性的软件只需要与 U_M 进行相似性比较，而不需要与整个恶意软件家族进行相似性比较。

同理，由 T_2 中所有的恶意行为特征对应的恶意软件构成的集合为 C_M，C_M 所包含的行为特征是整个恶意软件家族成员共有的。

如图 3 所示，将恶意软件行为特征量化后可得到特征轨迹，依据轨迹曲线可以发现具有“衍生”关系的软件的轨迹曲线会有交集，并且后代软件的特征节点的半径更大，即四元组分量的取值更大。因此，可以依据恶意软件家族内特征轨迹的重叠面积或交集数目、特征节点的半径或取值来生成家族的传递闭包关系，从而找到拥有整个软件家族恶意行为特征的恶意软件群 U_M 与拥有软件家族成员共有的恶意行为特征的恶意软件群 C_M。

4　基于高斯混合模型的增量聚类方法

4.1　传统的 GMM

基于模型的聚类方法试图优化给定数据和拟合某些数学模型。传统的基于模型的聚类方法 GMM 能够平滑地近似任意形状的密度分布[18]。GMM 使用无监督学习方法将数据划分为多个集群，每个数据簇由高斯分布近似，称为混合分量，具有自己的均值和协方差。假设一个 n 维样本空间中有随机向量 $\boldsymbol{x}$，若 $\boldsymbol{x}$ 服从多元高斯分布，则其概率密度函数为

$$N(\boldsymbol{x}\mid\boldsymbol{\mu},\boldsymbol{\Sigma})=\frac{1}{(2\pi)^{\frac{n}{2}}\mid\boldsymbol{\Sigma}\mid^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \quad (4)$$

其中，$\boldsymbol{\mu}$ 是 n 维均值向量，$\boldsymbol{\Sigma}$ 是 $n\times n$ 的协方差矩阵。高斯混合分布被定义为 k 个高斯分布的凸组合，如式(5)所示。

$$f(\boldsymbol{x}\mid\boldsymbol{\mu},\boldsymbol{\Sigma})=\sum_{j=1}^{k}\lambda_j N(\boldsymbol{x}\mid\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j) \tag{5}$$

其中，k 为混合成分的数量，$\boldsymbol{\mu}_j$、$\boldsymbol{\Sigma}_j$ 分别为第 j 个高斯成分的均值向量和协方差矩阵，λ_j 为混合系数，满足 $\sum_{j=1}^{k}\lambda_j=1$。

求解 GMM 时，需要使用最大似然估计来迭代更新 GMM 的参数，并计算该参数属于不同簇的概率，具体的求解步骤详见文献[19]。

传统的 GMM 通过假设待估计的分布来自固定成分数量的高斯混合分布，把密度估计问题转变为一个求解最大似然估计的问题，这样的假设大大减少了模型的参数，降低了空间复杂度。GMM 的学习过程从参数的初始估计开始，然后使用期望最大化（EM, expectation maximization）算法进行估计并最大化后验估计[20]。然而，GMM 对选定的初始参数估计过于敏感，并且可能会收敛到参数的边界，导致估计不准确。因此，在传统的高斯混合模型的基础上[21]，本文引入增量学习，构建基于增量的高斯混合模型来识别恶意软件家族。增量学习是指在学得模型后，接收到训练样本时，仅需根据新样例对模型进行更新，不必重新训练整个模型，不需要每次对所有数据进行重新聚类，这种学习方法很适合用来识别恶意软件家族。因为当需要检测一个未知的恶意样本时，只需要利用上一次聚类的结果，即已有的恶意软件家族，每次将一个数据样本划分到已有簇中或新增一个簇，这样新增的数据样本也不会影响原有划分。

4.2 基于高斯混合模型的增量构建

本文利用改进的高斯混合模型的树结构来刻画恶意软件家族内的传递闭包关系。一方面，传递闭包关系是依据恶意软件家族的进化史构建的，一个显著的特点是后代软件功能的强化与多样化；另一方面，GMM 树结构构建的依据是软件行为特征的一致多样性，根节点具有软件家族中最普遍的特征，叶子节点的特征最能代表整个软件家族的特征，而普通成员的特征需要与叶子节点的特征相似且比根节点的特征更具多样性。而软件的功能是软件外在行为的表现，软件的特征是对软件行为的标识，两者都是对软件行为的刻画。因此，本文提出的高斯混合模型的树结构能有效地揭示恶意软件家族的进化史及恶意软件行为的目的性、继承性与多样性。

Wan 等[11]提出的 ICGT 算法首先并未考虑节点的继承性与多样性，只是简单地将新的节点插入 GMM 树中，而且新的节点需要与所有的叶子节点进行简单的距离比较，简单的距离比较也不能反映节点的相似性。其次，当 GMM 树更新时，需要更新叶子节点的所有祖先节点的 GMM 参数，因此，ICGT 算法不仅不能揭示节点间的内在关系，而且数据的插入与更新速率太慢，影响聚类效率。本文依据恶意软件家族的目的性、继承性与多样性，提出了适用于恶意软件聚类的基于高斯混合模型的增量聚类方法，称为 ICGM（incremental clustering of GMM model）。ICGM 创建并动态调整与恶意软件家族的进化史相一致的 GMM 树，每一个 GMM 树代表一个有相同目的的恶意软件家族，由样本点生成的所有的 GMM 树就构成了 GMM 森林。GMM 树的定义为 $T_G=(N_R, N_M, N_L)$，其中，N_R 是根节点，由式(3)中 C_M 包含的数据样本构成，代表了家族成员共有的行为特征；N_M 是成员节点，是由 N_R 派生得到的；N_L 是叶子节点，叶子节点中的部分节点由式(3)中的 U_M 构成，代表恶意软件家族的恶意行为特征，本文将这群叶子节点记为 N_{LR}。

相对熵（KL, kullback leibler）可以用来度量 2 个概率分布之间的差异，当待比较的 2 个统计模型服从高斯分布时，KLD（kullback-leibler divergence）有闭式解[22]，如式(6)所示。其中，KLD(g_i,g_j)是指高斯分布 g_i 到 g_j 间的 KL 距离，n 是特征向量的维数。但由于 KL 不满足对称性，本文采用式(7)定义的距离度量方法来比较形如式(5)的高斯分布之间的相似性。

$$\mathrm{KLD}(g_i,g_j)=\frac{1}{2}\left[\ln\left(\frac{|\boldsymbol{\Sigma}_{g_i}|}{|\boldsymbol{\Sigma}_{g_j}|}\right)+\mathrm{tr}\left(\sum_{g_j}^{-1}\boldsymbol{\Sigma}_{g_i}\right)+(\boldsymbol{\mu}_{g_i}-\boldsymbol{\mu}_{g_j})^{\mathrm{T}}\sum_{g_j}^{-1}(\boldsymbol{\mu}_{g_i}-\boldsymbol{\mu}_{g_j})-n\right] \tag{6}$$

$$d(g_i,g_j)==\frac{\mathrm{KLD}(g_i,g_j)+\mathrm{KLD}(g_j,g_i)}{2} \tag{7}$$

由于 GMM 树的节点均是 GMM，当输入新的数据点时，需要度量数据点与 GMM 之间的相似性。本文给出的方法是，对于新的数据点也创建相应的高斯分布，其平均向量是数据向量，并赋予协方差很小的值，这样能使高斯分布被协方差约束在一个小的范围内。若新的节点对应的高斯分布为 g_{new}，

则 g_{new} 与拥有 k 个高斯分布的 GMM 之间的相似性度量如式(8)所示，其中，λ_l 为高斯分布的混合系数，g_l 为 GMM 的第 l 个高斯分布。

$$S(g_{\text{new}}, g_l) = \sum_{l=1}^{k} \lambda_l d(g_{\text{new}}, g_l) \tag{8}$$

基于高斯混合模型的增量构建算法如算法 1 所示。

算法 1　基于高斯混合模型的增量构建算法

输入　已知恶意软件家族数量 N，样本数量 M，样本 $X=\{X_1,\cdots,X_M\}\in N$，阈值 ϑ,ψ，j: $j\in[1,i_m]$, $i\in[1,N]$，其中，i_m 表示第 i 个 GMM 树的节点数

输出　插入 M 个样本点后的 GMM 树

初始化　GMM 树的根节点: $N_{R_i}=\varnothing$，i=1, j=1, k=1, S=0

1) 构建每个 GMM 树的根节点，根节点 N_{R_i} 由 C_M 中的数据点组成

2) while($i\leqslant N$ && $k\leqslant M$ && $X_k\neq\varnothing$)

3)　根据式(8)计算 GMM 树中数据点 X_k 和 N_{LR_i} 之间的相似性 $S(\text{gmm}_{X_k},\text{gmm}_{N_{\text{LR}_i}})$

4)　if ($S(\text{gmm}_{X_k},\text{gmm}_{N_{\text{LR}_i}})>\vartheta$)　then

5)　　$X_k\in \text{GMM}_i$

6)　　从根节点 N_{R_i} 开始，数据点 X_k 按照从上到下的顺序插入第 i 个 GMM 树

7)　　while($j\leqslant i_m$)

8)　　$S_{\text{bmp}}=S(\text{gmm}_{X_k},\text{gmm}_{N_j})$

9)　　if($S_{\text{bmp}}>S$)　then　$S=S_{\text{bmp}}$,$j_{\max}=j$

10)　　end if　$j=j+1$

11)　　end while

12)　　if($N_{j_{\max}}\notin N_{L_i}$)　then

13)　　update(X_k, GMM$_i$),$k=k+1$

14)　　else if ($S(\text{gmm}_{X_k},\text{gmm}_{N_{\text{LR}_i}})>\psi$) then

15)　　update(X_k, GMM$_i$),$k=k+1$

16)　　end if

17)　　否则，将生成一个新的叶子节点。该叶子节点仅有一个数据点 X_k，并且叶子节点的父节点是当前节点。令 $k=k+1$

18)　　end if

19)　　else

20)　　$X_k\notin \text{GMM}_i$, $i=i+1$

21)　end if

22) end while

23) if($X_k\neq\varnothing$)　then

24)　数据点 X_k 不属于当前任何恶意软件家族

25)　update($X_k,\varnothing$),$N=N+1$

26) end if

算法 1 采用基于高斯混合模型的方法，在构建增量的同时进行恶意样本聚类，最终通过恶意软件家族逻辑行为的相似性识别出 M 个恶意软件样本。当输入新的样本数据 X_k 时，仅需要与 GMM 树的 N_{LR} 进行相似性比较，若相似性满足阈值条件，则从根节点开始将新样本数据与 GMM 树的节点进行相似性比较，然后插入最优的节点中。否则，寻找下一个 GMM 树。当数据点 X_k 插入相应的 GMM 树后，需要更新 GMM 树。依据条件 1~条件 3，本文分 4 种情况讨论 GMM 树的更新方法，即 update 函数的实现。

条件 1　$S(\text{gmm}_{X_k},\text{gmm}_{N_{\text{LR}_i}})>\vartheta$，$i\in[1,N]$，$k\in[1,M]$。

条件 2　$N_{j_{\max}}\in N_{L_i}$。

条件 3　$S(\text{gmm}_{X_k},\text{gmm}_{N_{\text{LR}_i}})<\psi$。

下面，从恶意软件的继承性与多样性角度来分析 GMM 树更新的 4 种情况。

1) 若数据点 X_k 同时满足条件 1~条件 3，首先，数据点 X_k 继承了第 i 个恶意软件家族的目的性特征，则数据点 X_k 属于 GMM$_i$。并且数据点 X_k 需要插入 GMM$_i$ 的新叶子节点中，则将该叶子节点的高斯分布的平均向量赋值为 $\vec{X}_k$，并且协方差被赋予非常小的值。

2) 若数据点 X_k 满足条件 1 和条件 2，但不满足条件 3，则数据点 X_k 属于 GMM$_i$，且数据点 X_k 需要插入 GMM$_i$ 的已有叶子节点中，此时依据式(9)调整该叶子节点的混合高斯分布的参数即可。

3) 若数据点 X_k 不满足条件 1，说明 X_k 不属于当前任意一个恶意软件家族。此时，将节点的平均向量赋值为 $\vec{X}_k$，并且协方差被赋予非常小的值。令 $N=N+1$，找到一类新的恶意软件家族。

4) 当数据点 X_k 满足条件 1 但不满足条件 2，即数据点 X_k 需要插入 GMM$_i$ 的 $j_{\max}$ 节点中作为成员节点，说明数据点 X_k 不仅继承了第 i 个恶意软件家族的目的性特征，而且改进了家族的行为逻辑或者添加了新的恶意行为逻辑。此时，应依据式(9)调整节点 $j_{\max}$ 的混合高斯分布的参数。并且，依据家族的继承性可知，当前节点参数的变化会影响节点派生的子节点，

而派生的子节点应当拥有当前节点的所有行为特征，所以再根据式(9)，从节点 $j_{\max}$ 开始向下调整由 $j_{\max}$ 派生出的子节点的混合高斯分布的参数。调整完时，再依据式(1)重新确立第 i 个 GMM 树的 N_{RL_i}，从而保证聚类的有效性。同时，因为不需要对整个 GMM 树进行调整，提高了恶意行为的识别效率。

$$g^{(n+1)}(\boldsymbol{x}_{n+1}\mid\lambda_{n+1},\boldsymbol{\mu}_{n+1},\boldsymbol{\Sigma}_{n+1})=\frac{n}{n+1}g^{(n)}(\boldsymbol{x}_n\mid\lambda_n,\boldsymbol{\mu}_n,\boldsymbol{\Sigma}_n)+\frac{1}{n+1}g(\boldsymbol{x}\mid\boldsymbol{\mu},\boldsymbol{\Sigma}) \quad (9)$$

其中，n 表示在插入新数据点之前，当前节点中数据点的数量；$\boldsymbol{x}$ 表示插入的数据点向量；$\boldsymbol{\mu}$、$\boldsymbol{\Sigma}$ 表示插入的数据点的平均向量与协方差矩阵；$\boldsymbol{x}_n$ 与 $\boldsymbol{x}_{n+1}$ 表示插入前后的数据点向量；λ_n、λ_{n+1}、$\boldsymbol{\mu}_n$、$\boldsymbol{\mu}_{n+1}$、$\boldsymbol{\Sigma}_n$、$\boldsymbol{\Sigma}_{n+1}$ 分别表示插入新数据点前后节点的混合系数、平均向量和协方差矩阵。

传统聚类方法不能利用上一次的聚类结果，从而导致耗时、资源浪费等问题，而本文提出的 ICGM 算法引入增量学习，在接收到训练样本时，仅需根据新样本对模型进行更新，不必重新训练整个模型。当比较相似性时，新的样本点仅需要与恶意软件家族的部分叶子节点进行比较，并且当新的样本点插入 GMM 树后，仅需更新被插入节点的后代节点的高斯分布混合参数。因此，本文提出的 ICGM 方法适用于具有目的性、继承性与多样性的恶意软件家族的聚类。

5 实验分析

本文实验平台的具体配置如下，CPU 是 Intel(R) Core(TM) i5 2.50 GHz，内存是 8 GB，操作系统是 Windows10。为了获取样本的动态 API 函数调用序列，所有样本运行在一台主机上，具体配置如下，CPU 是 Intel(R) Core(TM) i5 2.50 GHz，内存是 1 GB，操作系统是 Windows XP Professional。实验框架如图 4 所示，共分为三大模块：恶意软件行为分析模块、恶意软件家族传递闭包关系构建模块和恶意软件识别算法模块。

在恶意软件分析模块，本文将分别提取样本的静态特征和动态特征，最后组成混合行为特征。提取样本的静态特征时，首先对样本进行静态分析，然后提取样本导入表中的 API 函数，以此作为静态特征；提取样本的动态特征时，首先通过动态分析，提取样本的 API 调用序列、参数和返回值；同时利用污点传播分析技术，对污染源进行标记，标记污点，并记录污点的传播路径，以此作为动态特征。然后将静态特征和动态特征进行组合，最后得到样本的混合行为特征。

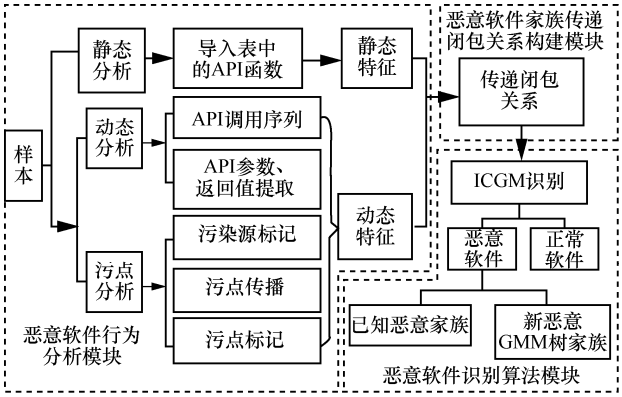


图 4 实验框架

在恶意软件家族传递闭包关系构建模块，首先依据每个样本的混合行为特征构建 L 个四元组，并做出软件在目的空间内的行为特征轨迹图。然后依据特征轨迹图的重叠面积或交集数目、特征节点的半径或取值来生成家族的传递闭包关系，从而找到拥有整个软件家族恶意行为特征的恶意软件群 U_M 与拥有软件家族成员共有的恶意行为特征的恶意软件群 C_M。最后利用算法 1 同时进行恶意软件家族的识别与恶意样本的聚类。首先判断是否为恶意软件，若不是，则标记为正常软件；若是，则进一步识别样本是否属于已知的恶意家族，若属于，则将其插入相应恶意家族的节点中，否则生成新的恶意家族 GMM 树。

5.1 实验准备

实验使用的数据集[23]如表 4 所示，包含了来自 8 个家族共计 10 826 个恶意软件样本。将每个家族的样本分为两部分，一部分用作训练，另一部分用作测试。训练集包括 1 600 个恶意样本和 2 800 个良性样本，测试集包括 9 226 个恶意样本和 10 000 个良性样本。

表 4 实验数据集描述

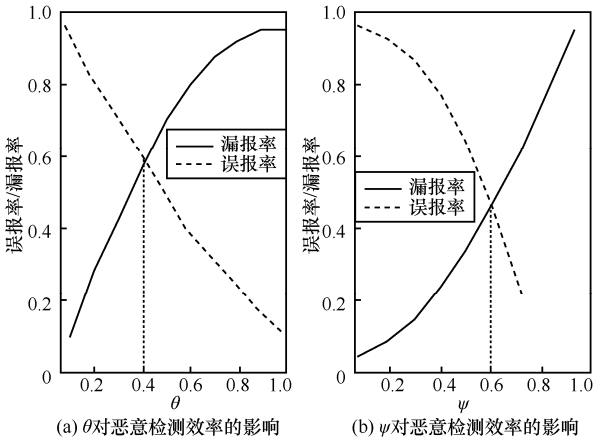
家族	样本总数/个	训练集/个	测试集/个
Kelihos_verl	398	200	198
Kelihos_ver3	2 942	200	2 742
Obfuscator	1 228	200	1 028
Ramnit	1 541	200	1 341
Vundo	475	200	275
Lollopop	2 478	200	2 278
Tracur	751	200	551
Gatak	1 013	200	813

在实验过程中，未知样本点通过本文提出的

ICGM 法将被识别为良性样本、已知家族中的恶意样本、未知家族的恶意样本。并通过以下参数来评价该方法的有效性：召回率或真阳性率（TPR, true positive rate）、误报率（FNR, false negative rate）、精确度和灵敏度的调和平均（F_1）、准确度（ACC, accuracy）。其中，TPN（false negative number）表示被正确判定为良性的样本点，TNN（true negative number）表示被正确判定为恶意的样本点（包括已知家族的恶意样本点与未知家族的恶意样本点），FPN（false positive number）表示 3 种类型的样本点：被错误判定为良性的恶意样本点、未知家族中被判定为已知家族的恶意样本点、已知家族中被判定为未知家族的恶意样本点，FNN（false negative number）表示被判定为恶意的良性样本点。则 TPR、TNR、F_1、ACC 的定义分别为

$$\begin{aligned}\mathrm{TPR}&=\frac{\mathrm{TPN}}{\mathrm{TPN+FNN}}\\ \mathrm{FNR}&=\frac{\mathrm{FNN}}{\mathrm{TPN+FNN}}\\ F_1&=\frac{2\mathrm{TPN}}{2\mathrm{TPN+FNN+FPN}}\\ \mathrm{ACC}&=\frac{\mathrm{TPN+TNN}}{\mathrm{TPN+TNN+FNN+FPN}}\end{aligned}\tag{10}$$

本实验中需要确定的参数有四元组的长度 L 和取值为(0,1)范围的阈值参数 θ、ψ。依据样本中多数恶意软件的行为，本文将 L 设置为 8。阈值参数的取值取决于参数对恶意检测误报率与漏报率的影响。图 5 仿真了阈值参数 θ、ψ 在(0,1)内取值时对恶意检测效率的影响，当 θ 取值范围为(0,0.4]时，漏报率低但误报率高；当 θ 取值范围为(0.4,1)时，误报率低但漏报率高。为了综合考虑恶意检测的漏报率与误报率，本文设置阈值 θ=0.4。同理，设置阈值 ψ=0.6。



(a) θ对恶意检测效率的影响　　(b) ψ对恶意检测效率的影响

图 5　阈值参数的选择

5.2　对比实验

结合长短期记忆（LSTM, long short term memory）深度学习模型[24]和随机森林模型，Lu 等[25]提出了一种基于 API 调用统计特征的恶意软件检测体系结构（ASSCA, API-based sequence and statistics feature combined malware detection architecture），将传统的机器学习与递归神经网络结合以获得更好的分类性能。Santos 等[8]提出了一种表示依赖于操作码序列的恶意软件的方法（OSDM, opcode sequence as representation of executable for data-mining-based unknown malware detection），使用静态检测方来提取可执行文件的操作码序列频率从而检测和分类恶意软件。Kolosnjaji 等[6]构建了一个基于卷积和循环网络层的神经网络（NCLN, neural network based on convolutional and recurrent network layer），并通过动态特征提取的方式得到了一种分层特征提取架构，将基于 *n*-gram 的卷积与完整的顺序建模相结合来检测恶意软件。

表 5 将本文提出的方法 ICGM 与 ASSCA、OSDM 和 NCLN 就真阳性率（TPR）与误报率（FNR）进行比较。由表 5 可知，ICGM 与 ASSCA 方法的 TPR 几乎相同，这说明基于软件的 API 调用序列可以表示恶意软件家族的行为特征，从而识别恶意软件家族成员。此外，OSDM、NCLN 的误报率远高于 ICGM 的误报率，说明采用静态与动态检测相结合的方式能更好地区分良性样本与恶意样本。

表 5　ICGM 与 ASSCA、OSDM 和 NCLN 方法的性能比较

家族	ICGM		ASSCA		OSDM		NCLN	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
Kelihos_verl	90%	0	90%	2.4%	75%	8.3%	84%	9.1%
Kelihos_ver3	64%	0.4%	63%	1.4%	56%	4.6%	50%	6.5%
Obfuscator	89%	0.2%	89%	2.6%	68%	7.4%	72%	7.3%
Ramnit	84%	0%	84%	1.6%	75%	5.73%	79%	3.41%
Vundo	85%	0.6%	84%	3.8%	61%	6.52%	66%	7.2%
Lollopop	78%	0	76%	2.6%	48%	7.24%	53%	5.63%
Tracur	75%	1.0%	75%	4.5%	70%	9.3%	72%	8.6%
Gatak	84%	0	84%	2.8%	79%	8.2%	64%	7.8%

表 6 将 ICGM 与 ASSCA 就精确度和灵敏度的调和平均（F_1）与准确度（ACC）进行比较。从表 6

可以看出，ICGM 的整体性能高于 ASSCA，因为 ASSCA 方法对于 API 调用序列的提取仅关注恶意软件自身行为方面，忽略了恶意软件家族的继承性，但是 ICGM 方法可以有效利用同一恶意软件家族行为的相似性来识别恶意软件家族成员。ICGM 方法用传递闭包关系来刻画软件家族的恶意行为的继承性与衍生性，得到了 4 个由具有特定目的的恶意软件家族的行为构成的传递闭包关系，然后定义并找到了恶意软件家族中代表性的特征集合与恶意软件集合，并将代表性的恶意软件集合作为 GMM 树的叶子节点。当需要辨别新样本点时，仅需要将新样本点与恶意软件家族的部分叶子节点进行相似性比较，并且当样本点插入 GMM 树后，仅需更新被插入节点的后代节点的高斯分布混合参数。因此，采用 ICGM 方法可以提高恶意软件家族的检测效率。

表 6 ICGM 方法与 ASSCA 方法的 F_1 与 ACC 值的比较

家族	ICGM		ASSCA	
	F_1	ACC	F_1	ACC
Kelihos_verl	86%	92%	72.5%	81.8%
Kelihos_ver3	91.5%	94%	74%	82%
Obfuscator	89%	90.7%	78%	75%
Ramnit	90%	87.5%	69%	72%
Vundo	85%	84%	81%	71%
Lollopop	92%	86%	63%	69%
Tracur	84%	91%	76%	82%
Gatak	90%	93%	65%	74%

Ahmed 等[26]同样提出过利用污点传播技术来跟踪分析 Windows 操作系统运行时 API 的调用序列，并且运用机器学习算法来提高恶意软件检测的准确性，该恶意软件检测方法称为 STAM（spatio-temporal information in API calls with machine learning algorithm）。表 7 列举了 ICGM 与 STAM 的检测时间（T）与存储空间（S）。从表 7 可以看出，ICGM 方法的存储成本明显低于 STAM 方法的存储成本。因为 STAM 方法需要存储来自同一个恶意软件家族的所有训练样本的 API 调用序列，而 ICGM 只需要存储恶意软件家族叶子节点中训练样本的 API 调用序列。检测时间包括生成 API 调用序列的时间成本与样本训练的时间成本。STAM 方法的检测时间约为 ICGM 方法的 1.3 倍，因为在检测恶意软件时，ICGM 方法只需要利用上一次聚类的结果，每次将一个数据样本划分到已有 GMM 树中或新增 GMM 树，这样新增的数据样本也不会影响原有划分，并且样本只需要与 N_{LR} 进行相似性比较，而不需要与整个恶意软件家族进行对比。因此 ICGM 方法不仅能节省存储空间，还能减少检测时间。

综上，本文提出的基于高斯混合模型的增量聚类算法不仅能节省恶意软件检测的存储空间，还能提高恶意软件的检测准确率与识别效率。

表 7 ICGM 与 STAM 方法的检测时间与存储空间的比较

家族	ICGM		STAM	
	T/min	S/KB	T/min	S/KB
Kelihos_verl	0.9	46.3	1.2	52.5
Kelihos_ver3	2.9	91.5	3.4	93
Obfuscator	2.1	78	2.9	85
Ramnit	2.7	80	3.9	84
Vundo	1.2	50.6	1.5	72
Lollopop	3.1	90.3	4.5	86.2
Tracur	2.6	72	3.4	81
Gatak	2.9	78	3.7	85

6 结束语

为了提高恶意软件的检测准确率与识别效率，本文在高斯混合模型中引入增量机制，提出了基于高斯混合模型的增量聚类算法来识别恶意软件家族，同时执行家族识别与恶意软件聚类。首先，依据恶意软件的目的性与继承性，本文从行为检测的角度通过追踪 API 函数调用的逻辑规则来提取恶意软件的特征。然后，从恶意行为的角度为每个恶意软件家族构建 4 个行为传递闭包关系，并建立特征行为与恶意软件的一对一映射关系。最后，本文创建并动态调整与恶意软件家族的进化史相一致的 GMM 树，采用基于高斯混合模型的增量聚类方法来识别恶意软件家族。

参考文献：

[1] POTTER B, DAY G. The effectiveness of anti-malware tools[J]. Computer Fraud & Security, 2009(3): 12-13.

[2] KIM J Y, BU S J, CHO S B. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders[J]. Information Sciences, 2018(460): 83-102.

[3] PENG W, LI F, ZOU X, et al. Behavioral malware detection in delay

tolerant networks[J]. IEEE Transactions on Parallel and Distributed systems, 2014, 25(1): 53-63.

[4] PEKTAŞ A, ACARMAN T. Malware classification based on API calls and behaviour analysis[J]. IET Information Security, 2017, 12(2): 107-117.

[5] HAN L, ZHOU M, HAN S, et al. Targeting malware discrimination based on reversed association task[J]. Concurrency and Computation: Practice and Experience, 2018: e4922.

[6] KOLOSNJAJI B, ZARRAS A, WEBSTER G, et al. Deep learning for classification of malware system call sequences[C]//Australasian Joint Conference on Artificial Intelligence. Springer, 2016: 137-149.

[7] CHO I K, KIM T G, SHIM Y J, et al. Malware similarity analysis using API sequence alignments[J]. Journal of Internet Services and Information Security., 2014, 4(4): 103-114.

[8] SANTOS I, BREZO F, UGARTE-PEDRERO X, et al. Opcode sequences as representation of executables for data-mining-based unknown malware detection[J]. Information Sciences, 2013(231): 64-82.

[9] ARP D, SPREITZENBARTH M, HUBNER M, et al. DREBIN: effective and explainable detection of Android malware in your pocket[C]//NDSS. 2014: 23-26.

[10] XU K S, KLIGER M, HERO III A O. Adaptive evolutionary clustering[J]. Data Mining and Knowledge Discovery, 2014, 28(2): 304-336.

[11] WAN Y, LIU X, WU Y, et al. ICGT: a novel incremental clustering approach based on GMM tree[J]. Data & Knowledge Engineering, 2018(117): 71-86.

[12] PFEFFER A, CALL C, CHAMBERLAIN J. Malware analysis and attribution using genetic information[C]// 2012 7th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2012: 39-45.

[13] WU S, WANG P, LI X, et al. Effective detection of android malware based on the usage of data flow APIs and machine learning[J]. Information and Software Technology, 2016, 75: 17-25.

[14] DAS S, LIU Y, ZHANG W. Semantics-based online malware detection: towards efficient real-time protection against malware[J]. IEEE transactions on information forensics and security, 2016, 11(2): 289-302.

[15] ZHAO H, XU M, ZHENG N, et al. Malicious executables classification based on behavioral factor analysis[C]//International Conference on e-Education, e-Business, e-Management, and e-Learning. IEEE, 2010: 502-506.

[16] DENG Z, LLOYD H, XIA C, et al. Components of variation in female common cuckoo calls[J]. Behavioural Processes, 2018(158): 106-112.

[17] SARACINO A, SGANDURRA D, DINI G. Madam: effective and efficient behavior-based Android malware detection and prevention[J]. IEEE Transactions on Dependable and Secure Computing, 2018, 15(1): 83-97.

[18] BOULEMNADJEL A, HACHOUF F, KHARFOUCHI S. GMM estimation of 2D-RCA models with applications to texture image classification[J]. IEEE Transactions on Image Processing, 2016, 25(2): 528-539.

[19] ENGEL P M, HEINEN M R. Incremental learning of multivariate gaussian mixture models[C]//Brazilian Symposium on Artificial Intelligence. Springer, 2010: 82-91.

[20] SONG M Z, WANG H B. Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering[J]. Proceedings of SPIE-International Society for Optics and Photonics, 2005 (5803): 174-184.

[21] TANG Z, SHEN F, ZHAO J. Speaker recognition based on SOINN and incremental learning Gaussian mixture model[C]//The 2013 International Joint Conference on Neural Networks. IEEE, 2013: 1-6.

[22] LIU Y, PERRONNIN F. A similarity measure between unordered vector sets with application to image categorization[C]//2008 IEEE Conference on Computer Vision and Pattern Recognition. 2008: 24-26.

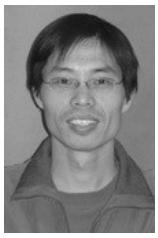
[23] RONEN R, RADU M, FEUERSTEIN C, et al. Microsoft malware classification challenge[J]. arXiv Preprint, arXiv: 1802.10135, 2018.

[24] LIPTON Z C, BERKOWITZ J, ELKAN C. A critical review of recurrent neural networks for sequence learning[J]. arXiv Preprint, arXiv: 1506.00019, 2015.

[25] LU X F, XIAO Z, JIANG F S, et al. ASSCA: API based sequence and statistics features combined malware detection architecture[J]. Procedia Computer Science, 2018, 129: 248-256.

[26] AHMED F, HAMEED H, SHAFIQ M Z, et al. Using spatio-temporal information in API calls with machine learning algorithms for malware detection[C]//The 2nd ACM Workshop on Security and Artificial Intelligence. ACM, 2009: 55-62.

[作者简介]



胡建伟（1973− ），男，浙江金华人，博士，西安电子科技大学副教授，主要研究方向为计算机网络、工业控制系统、网络软硬件设备的安全与攻防对抗等。

车欣（1993− ），男，安徽芜湖人，西安电子科技大学硕士生，主要研究方向为信息安全、工控安全等。

周漫（1994− ），女，湖北孝感人，华中科技大学博士生，主要研究方向为恶意代码检测、入侵检测、物联网安全等。

崔艳鹏（1978− ），女，吉林长春人，博士，西安电子科技大学副教授，主要研究方向为电子战信号处理、电子战系统模拟、雷达目标识别等。