第38卷 第13期
Vol.38 No.13

计 算 机 工 程
Computer Engineering

2012年7月
July 2012

• 专栏 •

文章编号：1000—3428(2012)13—0009—04 文献标识码：A 中图分类号：TP309

基于敏感 Native API 的恶意软件检测方法

白金荣[1,2]，王俊峰[2]，赵宗渠[2]，刘达富[2]

(1. 玉溪师范学院信息技术工程学院，云南 玉溪 653100；2. 四川大学计算机学院，成都 610065)

摘 要：分析恶意软件传播与破坏的行为特征，包括进程、特权、内存操作、注册表、文件和网络等行为。这些行为通过调用相应的 API 函数来实现，为此，提出一种基于敏感 Native API 调用频率的恶意软件检测方法，采用 Xen 进行二次开发，设计对恶意软件透明的分析监测环境。实验结果表明，使用敏感 Native API 调用频率能够有效地检测多种未知恶意软件。

关键词：恶意软件；API 调用频率；数据挖掘；动态检测；硬件虚拟

Malware Detection Method Based on Sensitive Native API

BAI Jin-rong[1,2], WANG Jun-feng[2], ZHAO Zong-qu[2], LIU Da-fu[2]

(1. School of Information Technology and Engineering, Yuxi Normal University, Yuxi 653100, China;
2. College of Computer Science, Sichuan University, Chengdu 610065, China)

【Abstract】This paper analyzes the malware behaviors of propagation and damage, including process, privilege, memory, registry, file and network. The malware accomplishes these behaviors by calling the corresponding Application Programming Interface(API) functions. It proposes a dynamic malware detection method based on perception of sensitive native API calls frequency. It develops transparent monitoring and analysis environment based on source code of Xen. Experimental results indicate that the method can identify unknown malware.

【Key words】malware; Application Programming Interface(API) call frequency; data mining; dynamic detection; hardware virtualization

DOI: 10.3969/j.issn.1000-3428.2012.13.003

1 概述

由于病毒、木马、蠕虫等恶意软件的泛滥，被感染的计算机日益增加，因此恶意软件已成为互联网的最大安全威胁。传统的基于特征码的检测方法只能检测已经被专业人员识别出特征的恶意软件，几乎不能检测未知的恶意软件，且原先能检测出的恶意软件经过加壳、混淆处理后又会无法检测。

为解决该问题，人们提出了启发式分析检测方法。启发式方法指任何利用规则和模式来检测未知恶意软件的方法，分为静态检测方法和动态检测方法 2 类。静态检测方法通过分析恶意软件的静态文件结构、二进制字节码、反汇编后的代码、反汇编后的静态系统调用等获取恶意软件的特征，利用分类算法在正常软件与恶意软件之间建立较好的分割线，实现已知和未知恶意软件的检测。基于静态的恶意软件检测通常容易受加壳、变形、多态技术的影响，同时静态检测方法没有真实地运行软件，判断是否为恶意的软件行为没有展现，静态检测过的软件，其行为不一定安全。动态检测方法的主要原理是将目标程序放置在一个沙盒模型中，通过监控目标程序运行过程的行为来判断是否为恶意程序。动态检测方法通常分为粗粒度方法和细粒度方法。粗粒度方法通过运行恶意软件分析其行为所对应 API(Application Programming Interface)调用序列来进行恶意软件检测，细粒度方法通过恶意软件的运行时动态指令序列来进行检测。

利用 API 调用序列和机器学习的方法来实现恶意软件检测是近年来信息安全领域的一个研究热点。文献[1]提出用系统调用短序列描述程序行为，并通过建立正常行为模型来判断 UNIX 进程是正常的还是恶意的。文献[2]提出利用 API 的参数信息和返回值的信息来进行恶意软件的检测。文献[3]对基于系统调用踪迹的恶意行为规范的自动生成进行了研究，主要针对恶意规范自动生成方法最坏复杂度较高的问题，给出了序列生成操作依赖图的策略和方法。文献[4]提出一种基于语义的恶意代码行为特征提取及检测方法，通过结合指令层的污点传播分析与行为层的语义分析，提取恶意代码的关键行为及行为间的依赖关系。文献[5]把 API 调用分为 7 个大类，单独考虑每个大类的 API 调用序列进行分类，再组合多个分类器，取得了较高的检测率。文献[6]设计并实现了一种结合虚拟机技术和 Windows 操作系统自身所具有的调试功能来获取恶意代码行为的模块。

以上方法主要通过 API 调用系列进行恶意软件检测，没有考虑 API 调用的频率信息。本文对恶意软件(病毒、蠕虫、木马)进行动态分析，发现恶意软件的传播和破坏主要表现在 6 个方面的行为特征：进程行为，特权行为，内存操作行为，注册表行为，文件行为和网络行为，这些行为的完成都是通过调用相应的 API 函数来实现。本文通过监测每个程序(恶意软件或正常软件)的 API 调用序列，统计每个程序以上 6 个行为方面相关 API 函数调用的频率，以此为特征通过数据挖掘的分类算法来训练学习，训练好的分类器能判别未知程序是否是恶意软件。

基金项目：国家“863”计划基金资助项目(2008AA01Z208)；四川省青年科技基金资助项目(09ZQ026-028)

作者简介：白金荣(1977－)，男，讲师、博士研究生，主研方向：网络安全，数据挖掘；王俊峰，研究员、博士、博士生导师；赵宗渠、刘达富，博士研究生

收稿日期：2011-07-22 **E-mail：**baijr223@163.com

2 Win32 API 调用机制

Windows 与其他许多操作系统一样通过硬件机制实现了核心态(Kernel Mode)以及用户态(User Mode)2 个特权级别。Win32 API 调用机制如图 1 所示。用户态应用程序通过调用 Win32 API 函数来请求基本的系统服务，Win32 API 主要函数封装在 Kernel32.dll 和 Advapi32.dll 中。而窗口管理和绘图相关的系统服务则分别实现于 User32.dll 和 Gdi32.dll。Win32 API 函数对程序调用的参数进行检查确认，之后 Win32 API 函数调用 Ntdll.dll 中与之对应的 Native API 函数。

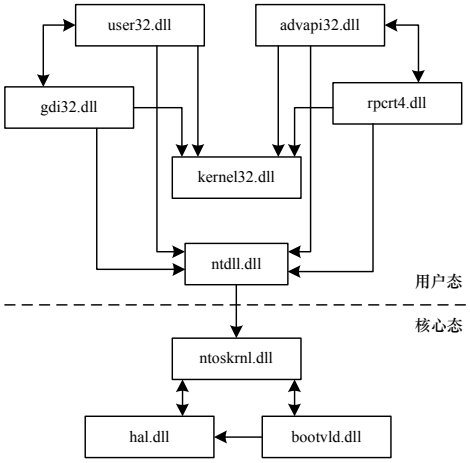

图 1 Win32 API 调用机制

Native API 是可由用户模式和核心模式程序调用的 Windows 系统服务集接口，它们直接由操作系统实现并在 Ntdll.dll 提供调用。例如，当应用程序调用 Win32 API CreateFile()时，最终会转换为对 Native API NtCreateFile()的调用。User32.dll、Aadvapi32.dll、Gdi32.dll、Rpcrt4.dll 以及 Kernel32.dll 等动态库里提供的 Win32 API 都是提供调用接口，没有具体实现，都要调用 Ntdll.dll 的 Native API 来实现相应功能。应用程序和内核程序可以直接调用 Ntdll.dll 里的 Native API 实现相应功能，如果要监控程序的 API 调用，最好直接监控 Native API 调用。如果是监控 Win32 API，程序绕过 User32.dll、Aadvapi32.dll、Gdi32.dll、Rpcrt4.dll 以及 Kernel32.dll 等动态库直接调用 Native API，就不能监控到。调用 Native API 在 Windows NT/2000 系统中，是通过软中断 Int 0x2E 实现调用，在 Intel x86 的 Windows XP/2003 系统中，处理器是通过执行 Sysenter 指令使系统陷入系统服务调用程序中，而在 AMD 的 Windows XP/2003 中使用 Syscall 指令来实现同样的功能。因此，要监视 Native API 的调用，同时监控 Sysenter、Syscall 指令和软中断 Int 0x2E。

基于程序行为的动态检测方法是目前反病毒技术的有效新途径。恶意软件与一般程序的区别主要在于执行了一些特殊的动作来传播和破坏系统。不管是二进制可执行病毒、脚本病毒还是宏病毒，它们都是一种程序，它需要调用操作系统提供的各种功能函数才能达到传播自身和破坏系统的目的。程序的执行流程和 API 调用序列在本质上是等价的[7]，通过监视程序所调用的 API 函数来实现对程序的行为监控是一种有效的方法。API 函数是微软提供给用户作为应用程序开发的接口，本身是不具备恶意性的。也就是说，恶意软件调用的 API，正常程序也是频繁调用。但综合考虑程序进程行为、特权行为、内存行为、注册表行为、文件行为和网络行为的相关 API 相同时间内的调用频率，恶意软件和正常程序间就具有很好的区分度了。本文通过监测每个程序的 API 调用系列，统计每个程序以上 6 个行为方面相关 API 函数调用的频率，以此为特征通过数据挖掘的方法来判别是否是恶意软件。

3 恶意软件检测模型

基于敏感 Native API 调用频率的恶意软件检测模型如图 2 所示。检测模型分为训练阶段和检测阶段 2 个阶段，训练阶段用于完成分类器的构建；而检测阶段用于完成恶意软件的检测。

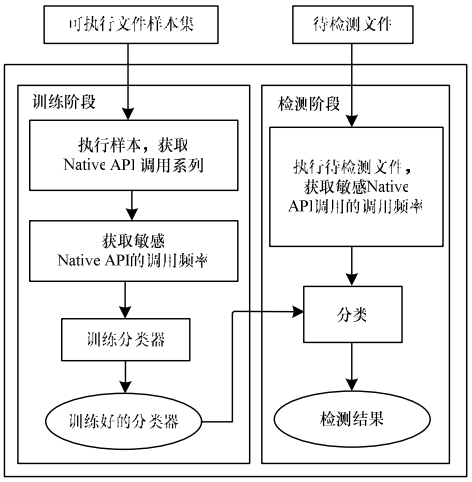

图 2 基于敏感 Native API 调用频率的恶意软件检测模型

模型由训练阶段的获取样本集的 Native API 系列开始，让样本文件在干净的分析环境中执行定长时间，记录下它的 Native API 系列。样本文件的 Native API 调用都比较多，很多 Native API 调用不能很好地区分恶意软件和正常文件，笔者基于对恶意软件行为的深入认识，把 6 种行为相关 Native API 调用提取出来，计算它在定长时间内的调用频率，然后使用这些特征来训练分类器，训练好的分类器能区分恶意软件和正常文件。

在检测阶段，把待检查文件放在干净的分析环境中执行，统计它在定长时间内的敏感 Native API 调用频率，使用训练阶段训练好的分类器，对待检测文件进行分类，得到是恶意软件或是正常文件的结果。

4 恶意软件行为分析环境

4.1 分析环境

专用的恶意软件分析环境有 4 种基本类型：基于真实系统来建立、基于虚拟机软件、基于 Bochs、Qemu 等系统级模拟器、基于 wine 等应用程序层虚拟技术。虽然这些分析环境得到了广泛的应用，但随着恶意软件的演化发展，部分恶意软件已经具备了反虚拟、反调试、反跟踪能力，使得这些分析环境部分存在以下问题：

(1)基于软件的虚拟能实现大部分虚拟工作，但少数指令的执行结果和在真实系统上执行的结果有差异，一部分恶意软件已经利用了这些特征来检测恶意软件是否在虚拟环境中执行，如果检测到在虚拟环境中执行，恶意软件就会改变执行路径，很难发现其恶意行为。此外，由于 x86 架构硬件的限制，基于软件虚拟的分析系统在宿主操作系统、虚拟机软件、被虚拟操作系统的特权等级分离不是很清晰，现在已经发现少部分恶意软件已可穿越虚拟机，破坏宿主操作系统，没有实现对被虚拟操作系统的绝对隔离。

(2)在真实系统上运行恶意软件是比较理想的一种环境，但也存在很多问题，比如环境恢复到干净状态比较费时，监

测单个恶意软件的执行很容易，但要批量的监测恶意软件的行为很难，因为执行一个恶意软件后都要把系统恢复到干净状态，程序实现上比较复杂。更大的问题是监测程序和恶意软件运行在同一系统内，这样很多恶意软件能检测到这种监控，恶意软件就会改变执行路径，很难发现其恶意行为。

(3)在虚拟机、模拟器或在被监控相应事件的真实操作上执行恶意软件，执行时间都会是真实操作系统上执行时间的很多倍，恶意软件对某个代码片段的执行时间简单的进行比对，就能发现其行为处以被监控之中。

针对上面提到的问题，本文基于开源虚拟机软件 Xen 进行二次开发，实现了对恶意软件透明的分析检测环境。Xen 是一款基于 GPL 授权方式的开源虚拟机软件，可用于 Linux 内核的一种虚拟化技术。Xen 位于 OS 和硬件之间，为 OS 提供了虚拟硬件抽象层。含有 Xen 软件的系统通常包含 3 个核心模块 Hypervisor、Guest 内核、用户程序。Hypervisor 的任务是将 Guest 系统跑起来，这些 Guest 系统也称之为域(Domain)。Xen 启动后，首先要做的事情之一就是加载 Dom0 内核，也就是说 Dom0 是第 1 个启动的 Guest，而且拥有相对其他 Guest 更高的权限，主要通过 Dom0 管理 Xen。其他的 Domain 称之为 DomU，U 表示非特权(unprivileged)。Xen 运行的层级高于所有的 Guest 操作系统，所有的 Guest 操作系统对硬件的访问都要通过 Xen Hypervisor。从 Xen 的架构可以看出，让恶意软件在 DomU 操作系统(WinXP)中运行，要监控恶意软件的行为，有 2 种方式：(1)在 DomU 操作系统(WinXP)中编写监控程序记录恶意软件的行为，然后通过通信程序把结果传送出来；(2)对 Xen 的代码进行二次开发，在 Xen Hypervisor 中监控恶意软件的行为。

第 1 种方式存在的问题是恶意软件和监控程序同在 DomU 操作系统(WinXP)中运行，恶意软件可能会检测到被监视，此外，监控程序和 DomU 操作系统的虚拟会使恶意软件的运行时间是真实操作系统上运行的多倍。本文实现了第 2 种方式，具体架构如图 3 所示。

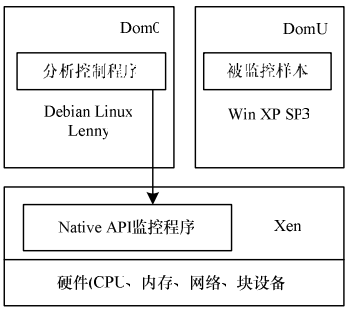

图 3　Native API 分析系统的架构

该分析平台具有以下 3 个方面的优点：

(1)基于硬件虚拟化技术，对 Xen、Dom0 操作系统、DomU 操作系统的特权等级进行了分离，恶意软件不可能穿越虚拟机。Xen 运行在 ring-1，被虚拟的操作系统(WinXP)运行在 ring 0，虚拟操作系统的应用程序运行在 ring 3。被虚拟的操作系统通过硬件虚拟直接在 CPU 上执行指令，和直接在硬件上执行指令一样，通过硬件虚拟指令保存 Xen、Guest 操作系统的状态和进行状态切换，Xen Hypervisor 对影响硬件状态的指令进行陷入处理，Xen Hypervisor 运行的特权等级比 Guest 操作系统高，在 DomU 中运行的恶意软件不可能发现被监视。

(2)实现了同时监控 Sysenter、Syscall 指令和软中断 Int 0x2E，保证了所有的 Native API 调用方式都处于监控之中。因为监控的是 Native API，保证了无论是从内核还是从用户态直接或间接调用 Native API 都处于监控之中，解决大部分方案只能监控从用户态调用 Win32 API 导致的漏监控缺陷。

(3)实现了被虚拟操作系统指令原始硬件执行时间的统计，以指令原始硬件执行时间来修改 DomU 操作系统的时钟，防止了恶意软件通过计算程序片段执行时间来检测是否被监控或是否在虚拟环境运行的问题。

4.2　监测控制程序实现

分析控制程序在运行在管理域 Dom0(Debian Linux Lenny)，恶意软件和正常文件样本在 DomU(被虚拟的 XP 操作系统)中运行。如果有要分析的样本，首先把 DomU(被虚拟操作系统 XP)恢复到干净状态，然后把 DomU 启动，把分析样本传送到 DomU 中，开启监控程序，使其监控样本的 Native API 调用，远程启动样本执行，开启定时器，如定时时间到，远程终止样本的执行，停止监控程序，关闭 DomU。如果要继续分析下一个样本，继续从开始执行，直到分析完所有样本。

监测控制程序伪代码如下：

```
procedure StartAnalazy();
repeat
获取样本文件
恢复被虚拟系统(Win XP)的映像文件为干净状态
启动被虚拟系统(Win XP)
把样本传人被虚拟系统(Win XP)
开启 Native API 监控程序
通过远程调用在被虚拟操作系统中运行传入的样本
Sleep(定长时间)
通过远程调用终止样本的执行
关闭 Native API 监控程序
关闭被虚拟系统(Win XP)
until 分析完所有样本
```

5　实验结果与分析

5.1　实验样本

实验样本分为恶意软件样本和正常文件样本。笔者从经过杀毒软件检测无病毒的 XP 系统 Windows 目录和 Program Files 目下获取了正常 PE 文件 350 个，从 vxheavens.com 网站下载了恶意软件 493 个，共计 843 个样本，恶意软件的分布如表 1 所示。

表 1　PE 格式的恶意软件分布

恶意软件类型	数量
Worm	137
Trojan	156
Virus	200
总数	493

5.2　评价方法与指标

因为实验是有限样本集实验，为了评估本文的检测方法，需要把样本分为训练样本和测试样本。交叉验证是统计学习中一个非常有用的实验方法，样本被随机均匀地分成 k 个子集，每个子集均做一次测试集，其余的作为训练集，重复 k 次，并将 k 次实验结果的平均作为结果。本文采用了最常用的 10 次交叉验证实验方法。

为了评价本文的检测方法，使用下面 3 个经典的评价指标：准确率，即所有被正确分类的数量在待测集合中所占比例；检测率，即被正确分类的恶意软件数目在待测集合的所

有恶意软件中所占比例；误报率，即被错误分类成恶意软件的正常软件在待测集合的所有正常软件中所占比例。

ROC 分析技术在最近几年越来越多的应用到机器学习领域中用来全面度量分类算法的性能，用于描述一个分类器在检测率和误报率之间的折中，能够直接比较多个分类器的性能。将 ROC 曲线描述的分类器性能转换为一个数值来表示分类器的性能，一个通用的方法是计算 ROC 曲线下的面积(Area Under the ROC, AUC)，0≤AUC≤1，AUC 的值越大，表示具有更高的检测率和更小误报率。

5.3 结果分析

分析监测每个实验样本 1 min，统计每个样本的敏感 Native API(进程行为 16 个，特权行为 36 个，内存行为 6 个，注册表行为 9 个，文件行为 5 个，网络行为 15 个)的调用频率，把敏感 Native API 的调用频率作为特征，使用 WEAK[8] 数据挖掘软件的 4 种分类算法 J48、RandomForest、Bagging(J48)、AdboostM1(J48)来训练 4 种分类器。训练后的 4 种分类器检测恶意软件的准确率差不多，都是 94%左右，可以任意选用一种。843 个样本进行 10 等份交叉验证的结果如表 2 所示。

表 2 恶意软件检测结果

分类算法	检测率/(%)	误报率/(%)	准确率/(%)	AUC
J48	96.8	10.0	93.90	0.952
RandomForest	98.0	11.7	93.90	0.969
Bagging(J48)	97.6	9.7	94.54	0.981
AdboostM1(J48)	96.6	7.4	94.90	0.979

4 种分类算法的 ROC 曲线如图 4 所示。

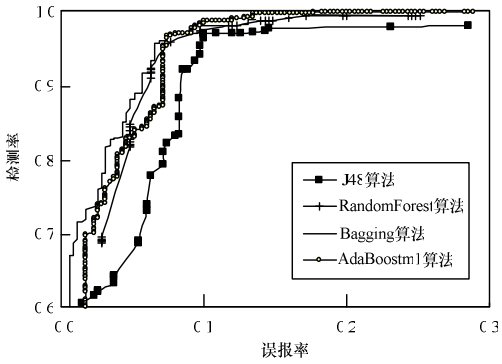

图 4 4 种分类算法的 ROC 曲线比较

可以看出，使用了集成学习方法(Boosting 和 Bagging)来改进 J48 的性能后，准确率有所提高。在检测率方面，恶意软件的检测率较高，正常文件的检测率稍微低一些。正常文件检测率低的原因是正常文件的样本大部分是 Win XP 的系统文件，这些文件中部分和恶意软件的特征非常相似，也较频繁地访问注册表、网络、文件、进程、内存和特权操作。

在 AUC 的性能方面，Bagging(J48)分类算法的 AUC 值是 0.981。可以发现，J48 比其他 3 种算法稍微差一些，这是因为其他 3 种算法采用组合多分类器提高了检测的性能。从图 4 的 ROC 曲线也可以明显看出，J48 分类算法也明显弱于其他分类算法。

恶意软件的性质决定了它的行为存在一些差异性，表 3 统计了正常文件、病毒、木马、蠕虫调用 6 种行为 Native API 的合计调用频率均值。从表 3 可以看出，在进程行为方面，正常软件的均值明显比恶意软件小，病毒与木马的相当，蠕虫的明显小，因为蠕虫主要通过网络方式攻击其他计算机。

表 3 各种行为的 Native API 调用频率均值 (次/分钟)

软件类型	每个样本的 Native API 调用频率均值					
	进程行为	特权行为	内存行为	注册表行为	文件行为	网络行为
正常文件	548.7	242.2	270.4	590.0	659.2	78.3
病毒	8 968.3	532.0	1 420.4	190.0	9 554.6	36.2
蠕虫	2 559.9	1 284.0	2 416.4	980.3	10 091.2	69.8
木马	9 832.6	925.2	630.0	2 903.5	14 889.0	122.0

在特权行为和内存行为方面，正常软件的均值明显比恶意软件的小，蠕虫的明显较大，因为蠕虫通过网络传播，感染后更多通过特权行为驻留系统，通过内存溢出攻击其他程序。在注册表访问行为方面，病毒的均值较小，因为病毒主要通过文件操作感染其他文件，它对注册表的操作主要是修改启动项，使病毒随系统同时启动；木马的均值较大，和它感染系统后要扫描注册表，窃取用户信息有关。

在文件行为方面，恶意软件的均值都较大，病毒的行为主要表现在读、写、创建文件，感染其他文件，蠕虫和木马的行为主要表现在遍历文件，读取有价值文件，窃取信息。考虑到运行恶意软件会通过网络传播，分析环境在和网络隔断的情况下进行，网络行为均值都较小，其中病毒的最小，因为病毒主要通过文件操作传播，正常文件的该值也相对较大，和很多系统文件是网络相关软件有关。

6 结束语

本文通过对常见恶意软件的动态行为分析，总结了恶意软件的行为表现，其在实现代码上表现为 Native API 的调用，为此，提出了基于敏感 Native API 调用频率的恶意软件检测方法。此外，针对现有恶意软件分析环境存在的问题，本文基于 Xen 二次开发了对恶意软件透明的分析监测环境。实验结果证明了本文方法的有效性。但本文只考虑了 Native API 的调用频率，没有考虑 Native API 调用的参数和返回值，同一 Native API 调用，参数不同有很大的差异性。因此，下一步将细化 Native API 调用，同时考虑调用参数和返回值的统计特征，进一步提高检测的准确率。

参考文献

[1] Forrest S, Hofmeyr S A, Somayaji A. A Sense of Self for UNIX Processes[C]//Proc. of IEEE Symposium on Security and Privacy. [S. l.]: IEEE Press, 1996.

[2] Mutz D, Valeur F, Kruegel C, et al. Anomalous System Call Detection[J]. ACM Trans. on Information and System Security, 2006, 9(1): 1-31.

[3] 孙晓妍, 祝跃飞, 黄 茜, 等. 基于系统调用踪迹的恶意行为规范生成[J]. 计算机应用, 2010, 30(7): 1767-1770.

[4] Forrest W S, Pearlmutter B. Detecting Intrusions Using System Calls: Alternative Data Models[C]//Proc. of IEEE Symposium on Security and Privacy. [S. l.]: IEEE Press, 1999.

[5] 王 蕊, 冯登国, 杨 轶, 等. 基于语义的恶意代码行为特征提取及检测方法[J]. 软件学报, 2012, 23(2): 378-393.

[6] Guo Shanqing, Yuan Qixia, Lin Fengbo, et al. A Malware Detection Algorithm Based on Multi-view Fusion[C]//Proc. of ICONIP'10. Sydney, Australia: [s. n.], 2010.

[7] 陈 培, 高 维. 恶意代码行为获取的研究与实现[J]. 计算机应用, 2009, 29(2): 76-82.

[8] Witten I H, Frank E, Hall M A. Data Mining: Practical Machine Learning Tools and Techniques[M]. [S. l.]: Morgan Kaufmann Publisher, 2011.

编辑 顾姣健