硕士学位论文

基于操作码序列的静态恶意代码检测方法的研究

A STUDY OF STATIC MALICIOUS CODE DETECTION METHOD BASED ON OPCODE SEQUNECES

卢占军

哈尔滨工业大学

2012 年 12 月

国内图书分类号：TP309.5　　学校代码：10213
国际图书分类号：004.49　　密级：公开

工学硕士学位论文

基于操作码序列的静态恶意代码检测方法的研究

硕士研究生：卢占军
导　　师：丁宇新 副教授
申请学位：工学硕士
学　　科：计算机科学与技术
所在单位：深圳研究生院
答辩日期：2012 年 12 月
授予学位单位：哈尔滨工业大学

Classified Index: TP309.5
U.D.C: 004.49

Dissertation for the Master Degree of Science in Engineering

A STUDY OF STATIC MALICIOUS CODE DETECTION METHOD BASED ON OPCODE SEQUENCES

Candidate: Lu Zhanjun
Supervisor: Associate Prof. Ding Yuxin
Academic Degree Applied for: Master of Science in Engineering
Specialty: Computer Science and Technology
Affiliation: Shenzhen Graduate School
Date of Defence: December, 2012
Degree-Conferring-Institution: Harbin Institute of Technology

摘 要

随着网络的普及、计算机技术的日益进步，如今计算机信息安全面临着很大的威胁，恶意代码是其中主要的攻击手段。恶意代码的增长及其技术的不断发展不仅会给人类的生活带来诸多不便，而且也会使用户及企业蒙受巨大的经济损失，有些甚至能危害到国家信息的安全。

随着恶意代码的检测技术和反检测技术的不断对抗发展，每天产生大量新的恶意代码给分析人员带来巨大的压力和严峻的挑战。传统的恶意代码检测能力已经远远不能满足需求。恶意代码检测技术分为静态和动态检测。静态检测不执行代码，通过代码的内容和结构信息实现检测；动态检测是在虚拟环境中通过代码执行的行为来判断。但是随着恶意代码混淆技术的发展，有些静态恶意代码检测方法受到了挑战，一些恶意代码通过隐藏自身的恶意行为来躲避在虚拟环境下的检测也使得动态检测技术无能为力。因此，如何应对恶意代码爆炸式增长，尤其是应对利用混淆技术产生的恶意代码的变种问题成为恶意代码检测技术研究的重点。

本文研究了基于操作码序列的静态恶意代码检测技术，与以往的静态检测技术不同的是本文提取了基于程序控制流程图的操作码序列作为恶意代码的特征。首先，对恶意代码进行基于信息熵的查壳和脱壳处理；其次，对脱壳后的恶意代码进行反汇编，通过编写插件来构建程序的控制流程图并提取操作码序列；再次，利用 n-gram 算法来提取操作码序列特征，并使用信息增益和文档频率的方法来选择特征；最后，使用 K 近邻、决策树及支持向量机三种机器学习分类方法实现恶意代码的检测。在实验中，通过选择特征的数量及其他方法来对比分析实验，实验结果通过正确率、误报率、漏报率三个指标进行评价。通过实验结果的对比和分析，本文提出的方法收到了很好的实验效果。

关键词：操作码；恶意代码检测；控制流程图

Abstract

With the wide spread of the network and the development progress of computer technology, computer information security is facing a great threat. The malicious code is the culprit. The growth of malicious code and the development of the technology, not only will bring much inconvenience to human life, but also make the enterprises and users suffer huge economic losses. Some can even harm the national information security.

With the development and confrontation of malicious code detection and anti-detection technology, the daily production of a large number of new malicious codes brings tremendous pressures to analysts. Today, malicious code detection capabilities have been far insufficient for demand. Malicious code detection technology is divided into two approaches, static and dynamic. The static detection gets the result based on the content and structure of the code, but the dynamic detection is by executing code in a virtual environment. However, with the development of code obfuscation techniques, some static detection methods are facing a challenge. Some malicious codes can hide their malicious behaviors to evade detection in virtual environment. Therefore, how to deal with the explosion of malicious code, especially in response to the variations of malicious code, becomes the focus of the research of malicious code detection technology.

In this paper, we propose a new method of static malicious code detection based on the opcode sequences. This method extracts the opcode sequences through the program's control flow graph as the characteristic, which is different from other methods. Firstly, shell malicious codes. Secondly, the disassembly of malicious codes, then build the program's control flow graph by writing plugin and extract opcode sequences. Thirdly, extract the characteristics using n-gram algorithm and select features using information gain and document frequency methods. Finally, detect the malicious code using machine learning classification algorithms such as K-Nearest Neighbor, Decision Tree and Support Vector Machine. In the experiment, we choose the different number of features and other methods to compare the experimental results based on the correct rate, the rate of false positive, the rate of false negative. It can come to a conclusion that our method is effective according to the comparison and analysis of the experimental results.

Keywords: opcode sequences, malicious code detection, control flow graph

目 录

摘 要 I
ABSTRACT II
第 1 章 绪 论 1
1.1 课题背景和意义 1
1.1.1 课题背景 1
1.1.2 课题意义 2
1.2 国内外研究现状 3
1.3 本文的主要内容与结构 5
1.3.1 本文的主要内容 5
1.3.2 本文的结构 6
第 2 章 恶意代码检测技术的研究 7
2.1 恶意代码简介 7
2.1.1 恶意代码的定义 7
2.1.2 恶意代码的分类 7
2.2 恶意代码的检测技术 9
2.3 本章小结 14
第 3 章 基于操作码序列的恶意代码检测方法 15
3.1 引言 15
3.2 数据预处理 15
3.2.1 查壳与脱壳 15
3.2.2 分析反汇编文本 17
3.3 提取操作码序列 19
3.3.1 构建控制流程图 19
3.3.2 提取操作码序列 22
3.4 操作码序列特征的分析 25
3.4.1 操作码特征的比较 25
3.4.2 操作码序列的分析 25
3.4.3 n-gram算法提取特征 26
3.4.4 信息增益 27
3.4.5 文档频率 28

3.5 恶意代码的分类 29
3.5.1 决策树 29
3.5.2 KNN 31
3.5.3 SVM 32
3.6 本章小结 33
第 4 章 系统设计与实验结果 34
4.1 系统设计 34
4.2 评价方法 35
4.3 实验数据 36
4.4 实验结果对比与分析 36
4.4.1 准确率比较 36
4.4.2 误报率比较 39
4.4.3 漏报率比较 41
4.4.4 实验结果分析 44
4.5 本章小结 44
结 论 45
参考文献 47
攻读硕士学位期间发表的论文及其它成果 51
哈尔滨工业大学学位论文原创性声明及使用授权说明 52
致 谢 53

第 1 章 绪 论

1.1 课题背景和意义

1.1.1 课题背景

随着计算机网络的快速发展，用户的信息分享和交流不仅仅构成了一个广泛的社交网络，也给人们带来了很多便利。然而，由于网络的开放性一些负面问题也不断涌现，如垃圾信息，恶意代码的传播等[1]。在利益的驱使下，有很多人为了某种目的会对网络进行攻击或者破坏，使得如今的计算机信息安全面临着很大的威胁，其中恶意代码是主要的攻击手段。恶意代码是指故意执行未经授权的有害行为的程序。通常，恶意软件分为病毒、后门、蠕虫、特洛伊木马、RootKit、脚本恶意代码、恶意 ActiveX 控件、广告软件和间谍软件等，它们中每一种类型都包含许多变种[2]。

在网络安全事件中，恶意代码的危害以及其带来的经济损失占很大比例，有些甚至会威胁到国家的安全。经过多年的发展，恶意代码的破坏性和种类都得到了增强，恶意代码经过网络传播对人们的日常生活影响越来越大。在商业利益的驱使下，恶意代码的发展迅速。

从八十年代初期开始，从 1981 年在苹果机上诞生了最早的引导区病毒 Elk Cloner 到 1986 年的第一个 PC 病毒 Brain Virus 的诞生，从 1988 年的第一个因特网病毒 Morris 蠕虫的顷刻间让 6000 多台计算机瘫痪到 1989 年第一个病毒家族的出现，著名的 Yankee 病毒曾在荷兰和英国造成了极大的恐慌。从 1989 年的第一个特洛伊木马 AIDS 出现到 1990 年的第一个多态病毒 Chameleon 的出现。从 1992 年第一个多态病毒生成器 Mte 的诞生到第一个病毒构造工具集、病毒创建库的生成。从 1998 年的 CIH 造成的数十万台计算机的破坏到 1999 年 Melissa、Happy99 病毒的爆发使得 E-mail 服务器和网络负载过重，到 2000 年“爱虫”及其变种病毒的爆发，感染了 4000 多万台计算机。从 2001 年的“红色代码”蠕虫在网络上大肆泛滥到 2003 年的 Slammer 蠕虫导致 90%脆弱主机受到入侵。从 2003 年“冲击波”蠕虫的爆发到 2004 年的“震荡波”蠕虫的疯狂传播。从 2004 年到 2006 年后门等恶意代码的不断涌现。从 2007 年轰动世界的“熊猫烧香”的大肆传播到 2008 年的“扫荡波”病毒的攻击。近年来，出现了“木马下载器”病毒，使得感染计算机后产生很多木马病毒导致系统瘫痪等。

从恶意代码的发展历史来看，无论其发展速度还是其危害性都在不断增强。为了避免许多杀毒软件检测一些恶意代码变得越来越复杂，从简单的 AppleII 病毒发展到复杂的内核病毒。恶意代码的传播机制也有很大的变化，从以前的被动式传播到现在的主动式传播。由于现在出现了很多恶意代码编制的工具，所以恶意代码的编制和发布速度很快，使得恶意代码的数量呈几何式增长。恶意代码的发展，给社会带来的经济损失是无法估量的，也给互联网用户带来很多不便，据统计有 80%以上的用户曾受到恶意代码的感染。所以，在互联网安全方面恶意代码问题是很关键的，也是很迫切需要解决的。

1.1.2 课题意义

在网络安全事件中，恶意代码的危害以及其带来的经济损失占很大比例。随着恶意代码检测技术和反检测技术的不断对抗发展，混淆技术成为反检测技术的很重要的手段。混淆技术又称代码迷惑技术主要通过垃圾代码的插入、指令重排序、变量重命名、等价代码替换及寄存器重新分配等方法来逃避检测[3]。由于编写新的恶意代码需要编写者有很高的技术，所以现实中涌现了很多恶意代码的编写工具。人们为了追求更多的利益，利用混淆技术使得恶意代码爆炸式增长。每天产生大量新的恶意代码给分析人员带来庞大的工作量和巨大的压力。传统的恶意代码检测能力已经远远不能满足反恶意代码技术的需求。如基于特征码扫描的恶意代码检测技术需要不断的更新病毒的特征库，这会使得特征库所占的系统空间不断增长，从而导致检测效率的逐渐下降。

恶意代码的不断增长不仅危害计算机网络，而且也给很多公司带来了巨大的经济损失，有些甚至会危及到国家信息的安全。恶意代码检测面临着严峻的挑战，需要不断的创新技术来应对，而恶意代码的增长主要是由混淆技术产生的变种导致，所以检测恶意代码的变种成为人们研究的焦点。恶意代码检测技术主要分为静态和动态两种检测方法。静态检测不执行代码，通过代码的内容和结构信息实现检测；动态检测是在虚拟环境中通过代码执行的行为来判断。现在使用最广泛的静态检测方法基于特征码扫描方法对于最简单的混淆技术都束手无策。基于二进制匹配的方法更是受到混淆技术的困惑。这种方法的缺点是特征描述不够泛化。动态的方法根据行为来判断程序是否恶意，但是一方面动态方法只能判断程序一次的执行路径，而程序的分支很多无法预知在某一次执行会是哪一条路径，这样的判断方法不够全面。另一方面，很多恶意代码会检测其所处环境是否是虚拟机环境，可以根据实际环境隐藏或者改变自己的行为来逃避检测[4]。根据上面的分析，本文提出了一种新的静态恶意代码检测方

法，通过结合了程序的控制流程图分析程序的操作码序列，这样既能弥补动态检测不全面的不足，也能克服混淆技术带来的困惑，为恶意代码检测技术的发展提供一个重要的思路。

1.2 国内外研究现状

自 1981 年第一个病毒 Apple II 出现以来[5]，国内外许多计算机安全的学者便投身于与恶意代码的对抗过程中。恶意代码编写技术的发展也推动了检测技术的发展，直到现在已经有很多恶意代码检测技术被广泛应用。

Sung[6]等人提出了基于系统调用的静态恶意代码检测方法，主要针对恶意代码的变种。该方法是将恶意代码反汇编并根据反汇编后的文本信息提取系统调用序列，并通过系统调用序列的相似度来判断。基于系统调用序列方法也可以用在动态检测过程中，在虚拟环境中执行恶意程序时可以提取执行时的系统调用序列，并使用 n-gram 算法来提取特征，然后进行分类。张波云[7]等人在虚拟环境中动态获得可执行文件的系统调用序列，并使用 n-gram 算法提取特征，使用粗糙集理论对特征降维并使用支持向量机实施分类。

为了解决混淆技术带来的困惑，一些学者研究基于程序的语义分析方法。语义分析是通过形式化抽象指令运行时的语义，通过符号执行[8]、模型检验[9]、逻辑推理证明等方法来分析程序的语义信息。Cousot.P 和 Cousot.R[10]提出了程序分析构造和逼近不动点语义理论，这为程序的语义分析提供了理论基础。M.Christodorescu[11]引入抽象模式库作为恶意行为自动机的符号，将恶意行为表示为带未解释符号的自动机，最后使用模型检验来实现检测。随后他提出一种基于语义的检测方法，用迹语义来描述恶意代码的行为，采用抽象解释方法检测恶意行为[12]。D.Preda[13]也借鉴了抽象解释的思想，证明了关于混淆技术产生的恶意代码检测的正确性和完备性。 Singh[14]通过分析反汇编文本的数据流信息，利用线性时态逻辑语义模型检测恶意行为。Kinder[15]分析了程序的控制流程图和函数之间的调用关系，用计算逻辑树描述恶意行为并公式化，最终使用模型检测方法检测。李佳静[16]等人提出了一种基于语义的行为分析方法，对函数调用及函数调用序列之间的依赖关系进行了详细的描述，该方法能准确描述恶意行为并有很好的泛化能力。用有穷自动机描述恶意行为，并引入数据流分析使用下推自动机描述程序的全局状态空间以提高分析精度，最终使用模型检测器实现检测。王晓洁和王海峰[17]提出一种基于语义模型匹配的检测算法，通过语义描述恶意代码的行为，这样对经过代码混淆技术处理的恶意代码的检测有很好的效果。

孔德光[18]等人提出一种结合语义的多态蠕虫的签名提取算法，提高了检测

的鲁棒性和准确性。G.Tahan[19]等人提出了一种新的自动签名提取算法，该算法主要针对恶意的可执行文件，被应用到高速恶意代码过滤装置中。Y.Tang[20]等人提出了一种利用多序列对比技术的简化的正则表达式签名算法，这种方法能产生更加准确的基于漏洞的签名。Y.Chen[21]等人提出了在网络层没有任何主机分析的蠕虫执行的脆弱性驱动的签名，实验效果非常好。现有的基于签名的恶意代码检测技术通过特殊的字符串特征来判断，其准确率非常高，但是其缺点是不能检测新出现的恶意代码，并且需要不断的更新特征库。现在大部分研究用基于 n-gram 序列的字节序列代替二进制特征码序列，这会提高分类的准确率。 Robert[22]等人提出了用操作码序列作为特征，然后使用文本分类的方法实现检测，并解决了数据不平衡问题[23]。

Schultz[24]等人第一次提出了应用数据挖掘模型来检测恶意代码，他们提取三种特征并使用不同的分类方法：程序的头文件信息，字符串信息，字节的序列，应用基于签名、基于规则的学习器 Pipper、朴素贝叶斯等方法进行分类。研究表明使用机器学习方法能提高准确率。后来 Kolter[25]使用 n-gram 算法提取字节序列作为特征，改进的决策树算法取得了很好的分类效果。

在参考文献[26]中，作者提出了使用 n-gram 算法提取特征后使用信息增益的方法来选择一些分类效果好的特征，并使用 K 近邻，基于 TFIDF 的分类器、朴素贝叶斯、支持向量机、决策树等分类方法，并取得了很好的实验效果。Kolterh 和 Maloof[25] 研究了恶意代码的家族的分类，基于恶意代码的功能行为，使用多分类方法将恶意代码分为蠕虫、木马、后门、病毒等，这更加细化了分类的结果，有助于对每一种恶意代码的研究，发现它们的共性，这也为以后的语义分析等方法奠定了基础。文献[27]中作者提出了一个层次特征选择的方法，即使用 n-gram 算法提取特征后选择那些出现频率高于某个阈值的特征，这种方法对于检测恶意代码的变种很有效。Raja[28]等人应用数据挖掘方法实现恶意代码的检测，他通过反汇编技术提取了恶意代码的操作码序列，使用了一种新的在文本分类领域的特征选择方法 CPD（Categorical Proportional Difference）。CPD 用来度量一个特征的区分能力，最终分类效果相对比较好。Dolev[29]提倡使用操作码来作为恶意代码的中间表示。操作码是机器语言的一个操作的一部分，它包含着指令的行为和程序的控制。近年来，操作码特征已经被用来检测蠕虫的变种和一些间谍软件[30]。将操作码提取出来作为标签，然后产生签名来判别恶意代码的变种。后来有些学者提取操作码并将其转化成操作码序列来检测未知的恶意代码[22]，实验使用三种分类算法取得了很好的实验效果。在文献[31]中，作者提出了使用变长的指令序列作为特征，并使用 Bagging

算法得到了很好的实验效果。也有人使用了十六进制码作为特征[32]。在恶意代码检测技术中使用操作码序列作为特征的研究相对还是比较少的，但是研究结果发现操作码序列是一种比较好的特征表示方法。

在文献[1]中，作者使用程序的控制流程图并用三种不同版本的黑客防御工具设计了一个分类算法，并取得了很好的实验效果。Ismail[33]提出了将程序控制流程图和函数调用拓扑用于将未知的恶意代码归类。使用函数调用拓扑的缺点是，攻击者能使用相似的函数调用或者改变函数调用的序列来逃避检测。Halvar[34] 利用程序控制流程的拓扑图的同构来实现检测。因为同一种族的恶意代码的拓扑图基本相似，这种方法也是适合检测恶意代码的变种。Igor[35]提出了一个新的检测未知恶意代码族的方法。该方法是基于操作码序列的出现的频率，并挖掘了每一个操作码序列的相关性。通过大量实验对比分析，该方法是非常有效的。Perdisci[36]等人提出了从 PE 文件提取一些特征，如标准和非标准部分的数目，可执行部分的数量以及 PE 头文件的熵信息，并使用不同的机器学习模型实现分类。后来他们开发了一个快速统计恶意代码的检测工具[37]。

综上所述，现有的恶意代码检测技术有很多，每一种方法都有自身的优缺点。这为后面的研究提供了基础的同时也带来了很多挑战。本文提出了一种新的恶意代码检测方法，结合了特征码、行为及机器学习的方法，提出了基于操作码序列的静态恶意代码检测方法，能更好的检测恶意代码。

1.3 本文的主要内容与结构

1.3.1 本文的主要内容

本文提出了一种新的静态恶意代码检测方法，主要是结合程序的控制流程图提取了恶意代码反汇编后的操作码序列。这种方法结合了基于特征码和行为的共同特点，最终使用机器学习的分类技术实现检测。这样既克服了动态检测的不完整性，又解决了混淆技术带来的困惑。通过对恶意代码的了解，首先，对恶意代码进行脱壳处理，因为一般的恶意程序为了逃避检测都采用加壳技术来隐藏其真实行为，为了在反汇编时能得到恶意代码的真实行为需要对其做脱壳处理。其次，对恶意代码进行反汇编，分析程序的结构和文本内容。然后，构建程序的控制流程图并提取每一条路径的操作码序列。接着，使用 n-gram 算法提取特征并使用信息增益和文档频率方法选择特征。最后，使用 K 近邻、支持向量机及决策树三种机器学习分类算法实施分类。本文的主要工作有以下几点：

第一：收集实验样本并对其做了查壳和脱壳处理并对脱壳后的恶意代码进行反汇编分析，然后提取基于文本的操作码序列。

第二：本文提出了一种新的基于操作码序列的静态恶意代码检测方法。结合程序的控制流程图提取操作码序列。这样既考虑到了程序的文本特征，也分析了程序的行为特征，为更好的检测恶意代码提供了保证。

第三：本文根据提出的新方法，在 vs2010 平台下结合 IDA SDK 开发了基于控制流程图和基于文本的操作码序列提取的两个插件。

第四：使用 n-gram 算法提取操作码序列特征，并使用信息增益、文档频率特征选择方法选择区分能力好的特征，最后使用 K 近邻、支持向量机及决策树三种机器学习分类算法实现了恶意代码检测系统。

1.3.2 本文的结构

本文共分为四章，全文的结构安排如下：

第一章是绪论，主要介绍了恶意代码的危害和发展及课题的研究意义，并详细阐述了国内外的研究现状。最后介绍本文研究内容及其结构安排。

第二章是恶意代码检测技术的研究，主要介绍恶意代码定义及分类和现有的一些恶意代码检测技术的原理及优缺点。

第三章是基于操作码序列的恶意代码检测方法，按步骤详细介绍了恶意代码的查壳和脱壳、反汇编技术、程序流程图的构造方法以及操作码序列的提取与选择。然后分析操作码序列特征及特征的提取与选择。最后简单介绍了三种机器学习分类算法。

第四章是系统设计与实验结果。详细介绍整个系统的设计和实验数据。然后通过准确率、误报率和漏报率三个指标对实验结果对比分析，并给出结论。

第 2 章 恶意代码检测技术的研究

2.1 恶意代码简介

2.1.1 恶意代码的定义

恶意代码（Malicious Code），是威胁网络安全的主要形式，关于它的定义有很多种说法。恶意代码可以从三个方面解释：首先，恶意代码是一段可执行的程序；其次，恶意代码的行为是恶意的，未经用户授权侵入计算机破坏系统或者窃取信息等；最后，有些恶意代码自动的在计算机之间传播，给网络的负载带来很大压力。Grimes 将其定义为：经过存储介质和网络传播，未经授权而去破坏计算机系统的程序或代码[38]。Ed Skoudis 认为，恶意代码为运行在计算机上的按照攻击者意愿执行的一组指令[39]。McGraw 指出恶意代码是故意破坏系统特定功能的代码[40]。恶意代码大部分是以二进制的可执行文件的形式存在。

2.1.2 恶意代码的分类

恶意代码根据不同的依据有很多种分类方法，没有一个统一的说法，但是常见的种类有：病毒、后门、蠕虫、特洛伊木马等，此外还有广告软件、恶意网页脚本、Rootkit、间谍软件等[41]。下面对几种恶意代码做简单介绍：

（1）病毒。早期恶意代码的主要形式就是病毒，病毒是指能够自我复制并将其嵌入被感染宿主程序，宿主文件一旦感染病毒就会感染其他文件。病毒有很好的潜伏性和破坏性，大量的病毒入侵系统会造成系统的瘫痪。病毒按照攻击平台可分为 DOS 病毒、MAC 病毒、Win32 病毒及 Unix 病毒等；按代码形式分为源码病毒、中间码病毒及目标码病毒等；按照宿主可以分为引导型病毒和文件型病毒。

（2）特洛伊木马。木马分为客户端和服务端，客户端安装在攻击者的主机上是控制端，服务端安装在受害者的机器上。木马可以使攻击者远程控制受害者的主机，造成受害者信息丢失等问题。木马有很好的隐蔽性，通过模仿正常的系统文件命名、与其他程序绑定、进程注入及拦截系统调用的方法伪装自己。木马也有很好的自启动性和自恢复性。常见木马有远程访问型木马、键盘记录型木马、密码发送型木马、FTP 型木马以及破坏型木马等。

（3）蠕虫。蠕虫是一种可以独立运行、自我复制及自动传播的恶意程序。

它通过网络、共享文件、电子邮件、移动存储设备以及有漏洞的主机等自我复制和传播。蠕虫的传播速度非常快，根据它的危害性可以简单分为无害型、消耗型和破坏型。无害型蠕虫感染主机后会产生很多垃圾文件减少系统的可用空间；消耗型蠕虫感染主机后，发送大量扫描数据包，消耗主机的 CPU 和内存资源，与此同时增加了网络的负载，降低网络的性能；破坏型蠕虫感染主机后会删除和破坏程序和文件，有时会泄露一些重要信息。

（4）后门。它是一种运行在目标系统中，能够绕过安全控制机制获得对系统的访问权，为攻击者提供通道的恶意代码。后门可以使攻击者远程控制目标主机，危害无穷。后门提供的通道有几种类型：本地权限提升、远程命令行访问、单命令远程执行、远程控制等。

（5）Rootkit。它是指帮助攻击者获取主机管理权限后，实现维持拥有管理权限的程序[42]。通常攻击者通过后门获取管理权限，并使用 Rootkit 维持管理权限使的恶意代码能隐藏在目标系统中。Rootkit 分为用户模式和内核模式。用户模式通过通道插入恶意代码、覆盖文件、API 钩子和 DLL 注入等方式达到目的。而内核模式通过安装恶意的设备驱动程序、打补丁、修改内存中运行的内核以及虚拟伪造系统的方式实现。

（6）间谍软件。它是在未授权的情况下窃取用户的信息并通过网络发送给攻击者的一种恶意代码。这种恶意代码不仅仅能泄露目标主机的数据信息，还可以提供恶意代码的植入接口使得被侵系统受到更加严重的破坏。

（7）广告软件。它是指在未经用户授权的情况下和别的程序捆绑在一起，以便经常弹出一些用户不想接受的广告。这种恶意程序目的是通过这种强制的方式做商业宣传。一些广告插件的安装会降低主机的性能。广告软件主要的危害是弹出一些色情或者恶意的广告，这会给用户带来很大的困扰。

（8）恶意网页脚本。它是指在网页中嵌入一些用脚本语言编写的有恶意行为的代码。当用户点击带恶意脚本的网站后，脚本通过修改目标系统的注册表、下载病毒或者加载木马程序等方式对被侵系统实施破坏行为。

下面对几种恶意代码做简单对比：

表 2-1 几种恶意代码的比较

名称	特征	存在形式	传播途径	传播方式	典型代表
病毒	自我复制、感染宿主	寄生在宿主文件	宿主程序的执行	半自动	Michelangelo 、CIH

表 2-1（续表）

名称	特征	存在形式	传播途径	传播方式	典型代表
木马	隐蔽性好，远程控制目标系统	独立个体	侵入目标系统	人工	Hydan Setiri
蠕虫	自我复制，自动传播	独立个体	网络、移动存储设备	自动	熊猫烧香
后门	绕过安全机制获得系统控制权	独立个体	网络、移动存储设备	人工	VNC NetCat
Rootkit	维护管理权限，隐藏文件	独立个体	网络、移动存储设备	人工	Adore FakeGINA
间谍软件	窃取信息	独立或插件形式	安装在目标系统	人工	Perfect Keylogger
广告软件	强行弹出广告信息	独立或插件形式	安装在目标系统	人工	PurityScan KeenValue
恶意网页脚本	在网页中加载一些木马、病毒等	嵌入网页	网页浏览	人工	一些恶意网站

2.2 恶意代码的检测技术

随着恶意代码检测技术和反检测技术的对抗发展，现在已经有了很多检测技术。恶意代码检测技术主要分为静态和动态检测。静态检测不执行代码，通过代码的内容和结构信息实现检测；动态检测是在虚拟环境中通过代码执行的行为来判断。下面将详细介绍几种检测技术。

（1）特征码检测方法。基于特征码的检测技术，是通过对恶意代码的文本内容进行分析，提取二进制、字符串、字节序列、文件名等特征，将这些特征码存入特征库。当检测样本时，通过扫描样本的相关特征和特征库进行匹配，若有匹配的特征则判定该样本与匹配的为同一类型的恶意样本。这种方法比较简单并且检测速度较快，是很多商业杀毒软件采取的主要方式。但是这种技术也有很多缺点：首先，特征提取时有的是自动生成，但是有一些需要该领域的

专家人工提取比较好的特征；其次，这种技术只能检测已有的恶意代码，对未知的恶意代码种类束手无策；再次，该技术提取的特征的泛化能力不足，很容易受到混淆技术的干扰；还有，这种方法使得特征库不断增加，这需要用户经常更新特征库，随着时间的流逝，特征库会越来越庞大，这会影响检测的速度和系统的性能。

从图 2-1 中可以看出，当恶意代码入侵时，首先扫面其特征与特征库匹配，如果检测该样本为恶意样本则将其清除，否则分析恶意代码并提取特征将其添加到特征库中，以便以后的检测。

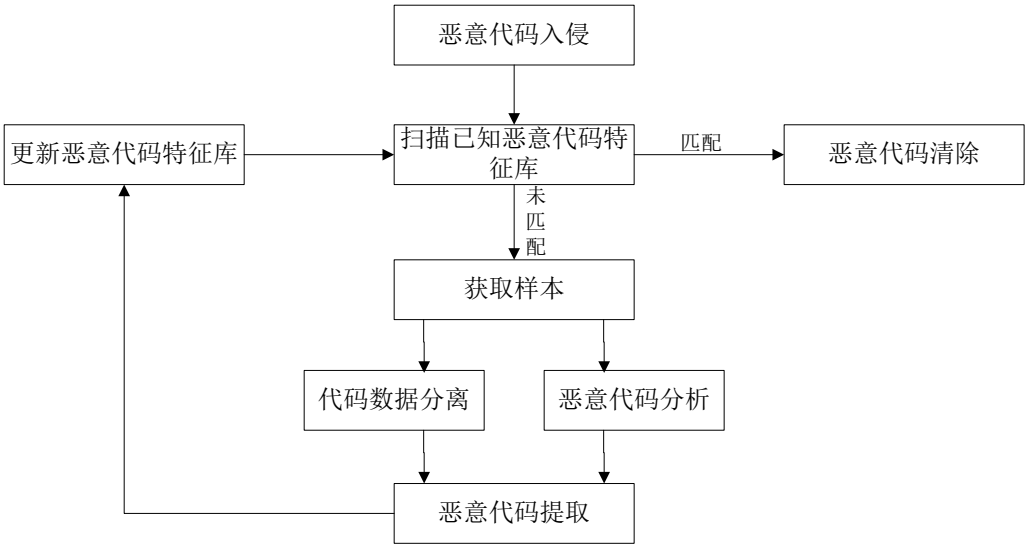


图 2-1 基于特征码方法检测的原理图

（2）启发式法。基于启发式检测方法的基本思想是通过分析恶意软件的特征和行为，定义一系列的规则，根据规则去进行判断。如最简单的方法是计算文件的风险值，根据经验给出一些行为的风险系数，通过统计设定一个风险阈值，然后根据程序的行为累加各种行为的风险值，当超过预先设定的风险阈值时则判定为恶意代码。例如，一些恶意代码执行时，会调用一些特定的内核函数，而往往这些函数的顺序也有规律，可以根据这些特征来指定一些规则检测同类恶意程序。这种方法也需要人工的参与分析，如果是自动提取也难免会受到混淆技术的干扰。

（3）校验和法。由于很多恶意代码寄生在被感染的文件中，当文件被感染后文件的属性大小和修改日期会被改变。基于检验和的检测方法（又称完整性验证法），就是基于这样的原理，通过 hash 或者别的方法记录文件原来的检验和，等到下次使用文件时首先检验该文件的检验和是否和原来的一致。这种方法可以识别未知的恶意代码，但是这种方法对文件的改变太敏感，容易提高误

报率，而且对一些隐藏性较好的恶意代码束手无策。

（4）权限控制法。基于权限控制法的方法是控制恶意代码在被侵入系统中的权限。因为恶意代码一般是可执行文件，在侵入系统只有执行程序才能实现其攻击的目的，而恶意代码要执行首先要获得可以执行的这个权限。一般的恶意程序入侵系统后会获得对系统的控制权，只有这样才能实现它的价值。所以通过控制权限将系统的权限降到刚好能完成宿主文件的正常功能。这样恶意代码的请求将得不到批准起到了防御的措施。但是这种方法的缺点是对于那些在正常权限下就能完成特定目的的恶意代码没有作用，此外这种方法只能起到防御作用，并不能真正检测出恶意代码的类别。

（5）专家系统。基于专家系统的检测方法是通过安全领域的很多专家对现有的恶意程序分析总结了很多特征、规则，并建立了相应的数据库。系统根据数据库的规则来检测样本。这样的方法简单有效，但是这会受限于数据库的完备性，需要很多专家为维护数据库付出很大代价。此外，这种方法也不能检测未知的恶意代码。

（6）行为法。基于行为的方法即动态的监控检测技术，通过将待检测的样本在虚拟机或沙箱中执行，根据程序执行的行为来判断程序是否是恶意的。在虚拟机和沙箱中运行恶意程序是为了防止恶意程序影响系统的正常运行。这种方法的好处是无论恶意程序加壳或者是通过混淆技术来变形都不会影响对其结果的判断，这种方法只注重分析程序的行为。而这种方法需要程序在虚拟机中运行，这会消耗时间和资源，不能保证检测的时效性。此外，有一些恶意代码能监测自身所处的环境，一旦检测到所处的环境是虚拟机便会执行一些其他的行为来躲避检测。对于这样的恶意代码动态检测技术无能为力。

（7）基于语义的检测方法。基于语义的检测技术是现在研究的热点。因为混淆技术只是通过插入垃圾指令、改变指令顺序及寄存器重新分配等方法来改变程序，但是程序的基本语义是等价的。通过分析恶意程序，抽象程序指令的行为并建立其行为模型，使得该模型既描述恶意程序的基本行为，又具有很强的泛化能力。这样因其有很强的泛化能力，使得检测恶意程序的变种更加方便快捷。除此之外，也可以检测未知类型的恶意代码。现阶段基于语义的检测方法分为基于内存和函数调用的方法。M. Christodorescu[11,12]提出了一套抽象理论和语义框架，使用自动机描述程序的行为，通过抽象理论描述程序的行为建立抽象模式库，并将其作为自动机的符号表，最终经恶意行为描述为自动机表述的模板，最后通过模型检测方法检测样本是否含有恶意行为。模型检测是通过遍历系统所有状态空间，看其中是否有一条符合的路径状态。之后，他还提出

迹语义这一概念，将迹语义作为程序的基本语义，并定义了等价的条件，通过抽象解释的方法给出了近似的检测算法。抽象解释理论为解决不可判定和复杂问题的逼近求解提供可很好的构造方法。基于函数调用的方法是将程序中使用的函数提取出来，并结合程序的控制流程图，通过图的同构、线性时态逻辑、计算逻辑树、有穷状态机及下推自动机等方法描述恶意行为，最终通过模型检测完成恶意代码的检测。

从图 2-2 中可以看出，首先，要根据语义分析的理论分析程序，并提取行为特征建立语义模型最终形成一个语义模板库。其次对于要检测的程序，对其反汇编分析程序，然后采用抽象理论方法中间表示程序的行为并建立模型。最后通过模型检测器实现检测。这种方法是有效的、是很重要的技术，不仅仅能检测变种，也能检测新的恶意代码种类。但是，这种技术实际操作起来比较困难，最主要的是程序的抽象表示和模型建立，这还需要后人不断的努力探索。

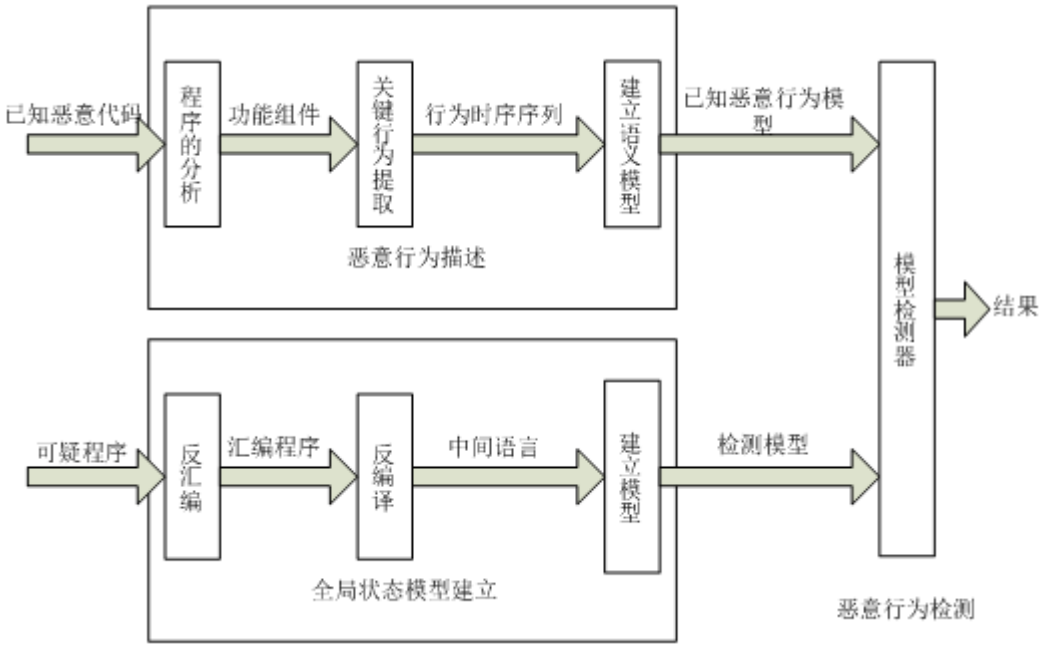


图 2-2 基于语义的恶意代码检测的模型

（8）基于机器学习和数据挖掘的检测方法。随着检测技术的不断发展，机器学习和数据挖掘的方法已经被开始应用在恶意代码检测的领域。主要应用分类、关联规则挖掘、序列模式分析以及聚类等几种技术。主要思想是利用数据挖掘技术从现有的数据中挖掘一些有意义的模式，用机器学习技术归纳出已有样本的特征，然后根据特征的相似性等完成分类的任务。其中，最主要的是选择好的特征和有效的分类器。检测步骤如下：首先，要分析样本确定提取哪种

特征或者特征序列；其次，根据特征的特性选择合适的特征选择方法从所有提取的特征中选择一些分类效果好的特征；最后，根据实际情况选择较好的模型实现分类。下图详细介绍了现在应用机器学习实现恶意代码检测的几种方法。

从图 2-3 中可以看出，现在已经有很多机器学习技术应用到了恶意代码检测这个领域。首先，在特征的表示方面有基于 n-gram 算法提取的字节序列、操作码序列、函数调用序列、可执行文件的特征和字符串等特征；其次，使用信息增益、文档频率、Fisher Score 及层次特征选择等方法选择分类能力较强的特征；最后采用人工神经网络、贝叶斯网络、朴素贝叶斯、决策树模型、K 近邻、支持向量机、随机森林、改进的决策树、改进的神经网络、VFI（Voting Feature Intervals）Hyper-Pipes 分类器以及 PART 分类器等实现分类。其中，Hyper-Pipes 分类器主要思想是记录训练样本的每一个属性和类别值的范围，当检测时选择检测样本与训练样本范围最近的作为分类结果。VFI 是通过将属性数字离散化构造点区间，记录每一个属性中类别的数量，分类时通过投票机制来实现。PART 分类器是结合 C4.5 和 PIPPER 算法开发，他的主要特点是克服了 C4.5 和 PIPPER 算法需要全局优化成合适规则的缺点[43]。此外，还有通过关联规则挖掘及序列模式挖掘等方法实现分类，这种方法是人们现阶段研究的热点。

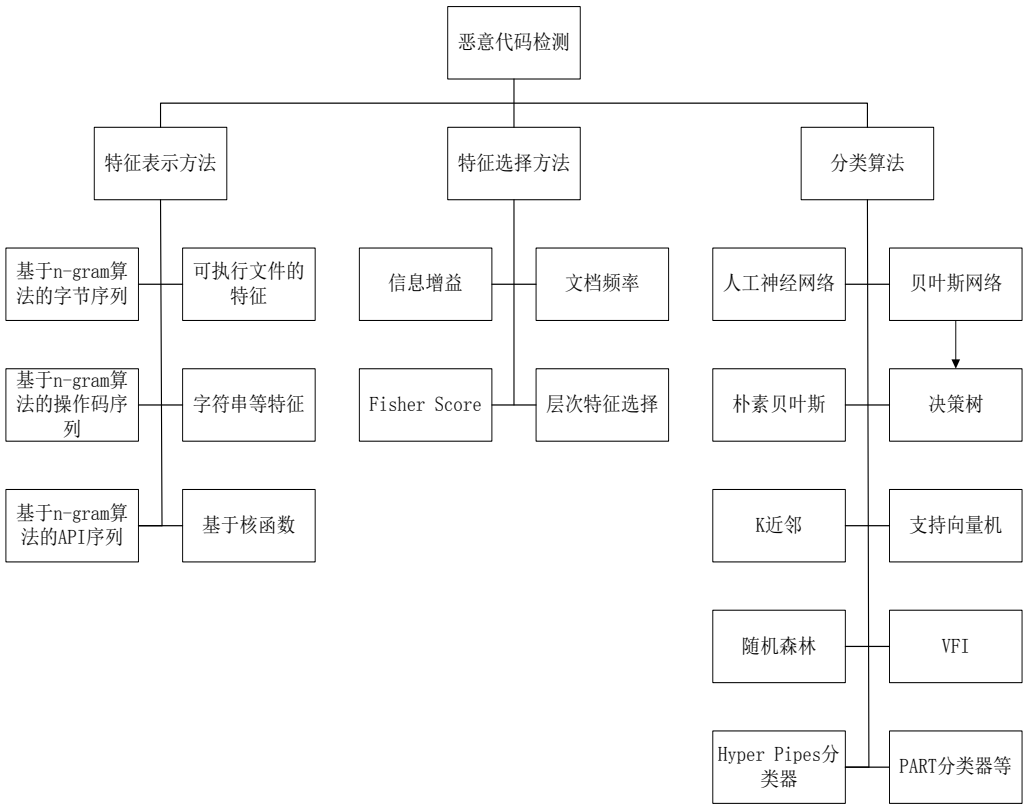


图 2-3 现有的基于机器学习的恶意代码检测方法的简单总结

2.3 本章小结

本章首先介绍了恶意代码的定义并详细介绍了恶意代码的分类，分别介绍了病毒、木马、蠕虫、后门、Rootkit、间谍软件、广告软件及网页恶意脚本等恶意代码的特征和危害。其次，将每一种分类通过从特征、传播方式、存在形式等方面做了相关的对比。最后，详细阐述了现有的几种恶意代码检测技术，分别介绍了基于特征码、启发式、校验和、权限控制、专家系统、行为、语义以及机器学习和数据挖掘的检测技术的原理及优缺点。

第 3 章 基于操作码序列的恶意代码检测方法

3.1 引言

本文研究了基于操作码序列的恶意代码检测方法，通过结合程序的控制流程图分析每条可能的执行路径的操作码序列来实现恶意代码的检测。研究方法的步骤是：首先，数据预处理，对恶意代码查壳和脱壳处理并对脱壳后的恶意代码进行反汇编分析；其次，按顺序提取基于文本内容的操作码的序列并提取基于控制流程图的操作码序列；然后，分析恶意代码和正常代码的操作码序列的特征，根据 n-gram 算法提取操作码序列特征并使用信息增益和文档频率两种特征选择方法选择分类效果较好的特征；最后，使用 K 近邻、决策树和支持向量机三种分类算法实现恶意代码的检测。下面将依次对每一个步骤详细介绍。

3.2 数据预处理

3.2.1 查壳与脱壳

为了隐藏其真实意图躲避杀毒软件的检测，或者是为了压缩体积便于传播和渗透，大部分恶意代码都经过加壳处理。但由于编写自己的加壳程序需要较深厚的技术水平，所以一般的恶意代码都是用现有的加壳工具加壳。当加壳后的程序执行时，首先获取壳程序执行时需要的 API 地址，然后解密原程序，最后重定位跳转到原程序入口点，把控制权转交给原程序。

为了很好的分析恶意代码原程序，首先要做的是查壳与脱壳工作。现阶段的查壳技术主要有两种：基于特征码和基于信息熵。

基于特征码：同样的加壳方法会使得加壳后的 PE 文件在特定的位置有相同的字节序列。相同的序列即为该加壳方式的特征，然而这种基于特征码的查壳方法，只能检测出已有的并且加入到特征库的加壳方法。

基于信息熵：经过加壳的 PE 文件，其结构会产生变化。壳一般分为加密壳和压缩壳，加密和压缩会使 PE 文件变成随机性更大的无结构形式。然而，熵是用来衡量不确定性的，这样经过加壳的 PE 文件的熵一般会大于源文件 PE 文件的熵。表 3-1 为本文选取的恶意代码样本加壳与脱壳后的熵值的比较，表中的结果都是通过 PEID 软件查壳和脱壳并计算其信息熵获得。从表 3-1 中看出，有几种加壳方式，有的是压缩壳，如 UPX、FSG 等；有的是加密壳，如 nSpack、ASPack 等。除了表中列出的壳名称，还有很多种壳，但是大致分为压缩壳和加

密壳。从表中可以直观的看到，加壳的恶意代码的信息熵均大于 7.5，而脱壳后信息熵均小于 6.2，本文对实验样本的信息熵做了统计，大部分加壳的恶意程序的信息熵大于 7，而不加壳的程序一般都小于 6.5，但是也有某些特别的样本。

表 3-1 恶意代码加壳和脱壳后熵值的对比

恶意代码名称	壳名称	加壳的熵值	脱壳后的熵值
Mydoom.a.exe	UPX	7.62	3.47
Mydoom.d.exe	UPX	7.88	5.69
Mydoom.e.exe	UPX	7.88	6.02
Mydoom.g.exe	UPX	7.89	5.66
Mydoom.i.exe	UPX	7.90	5.68
NetSky.c.exe	PEtitle	7.92	3.22
NetSky.d.exe	PEtitle	7.91	2.64
NetSky.e.exe	PEtitle	7.92	3.17
NetSky.f.exe	PE Pack	7.62	6.04
NetSky.k.exe	tElock	7.96	2.48
Nimda.b.exe	PC Shrinker	7.82	6.04
Fujack.k.exe	ASPack	7.70	6.12
Fujack.r.exe	nSpack	7.95	5.82
Viking.dz.exe	FSG	7.90	6.25
Alcoula.a.exe	UPX	7.62	2.59

目前有很多加壳工具，常用的有 ASPACK、PE PACK、UPX、PECOMPACT 等。相应的也有一些查壳软件有 PEID、FILEINFO、FILE SCANNER 等，本文主要是用 PEID 的批量模式查壳。

脱壳与加壳是相反的过程，为了更好的分析恶意程序的真实意图，首先要去掉外面的保护壳，找到原程序的真正入口点。脱壳分为手动和自动。手动脱壳需要脱壳者有很深厚的技术水平，跟踪和分析加壳的方法，然后再对应的去脱壳。而自动脱壳是运用专门的脱壳工具进行脱壳处理。随着逆向技术的发展，产生了大量的脱壳软件，有的是专门脱某一种壳，也有的能脱几种壳，然而这些软件都是只能脱掉现有的壳，对于一些新的壳还需要手工脱壳。目前常用的脱壳软件有 WASPACK、UNDBPE、UNFSG、EUNPACKER、VMUNPACKER 等。本文使用的脱壳工具是 VMUNPACKER。

3.2.2 分析反汇编文本

反汇编技术是一项非常重要的逆向技术，反汇编是为了在没有原代码的情况下对程序的行为了解。恶意软件的作者很少会提供他们“作品”的源代码，这就使得人们不得不对二进制的可执行代码进行反汇编。反汇编是指在对可执行代码的二进制文件进行分析得到汇编代码程序，通过分析汇编代码获得程序行为的方法。基本的反汇编算法有两种：线性扫描反汇编和递归下降反汇编。

线性扫描反汇编算法从代码段的第一个字节开始，以线性模式逐条反汇编每条指令。其优点在于：它能够扫描所有代码段。缺点：对数据和代码混合的情况没有考虑。

递归下降反汇编算法重视控制流的概念，通过对可执行代码的扫描来获得比较准确的反汇编结果，根据每条指令的引用来进行反汇编。其优点在于：它能将代码段与数据段区分开,可以有效的跳过嵌入在程序段中的数据及其无效的代码。缺点在于：它无法处理间接代码路径[44]。

反汇编技术非常重要，应用也越来越广泛，随之也产生了很多反汇编的工具，主要有 IDA Pro、W32Dasm、C32Asm 等。本文使用了强大的反汇编工具 IDA Pro，简称 IDA。IDA 采用递归下降反汇编算法并结合启发式技术来弥补递归下降算法的不足。它提供了各种 API 接口以及 SDK，使用 SDK，用户可以创建加载器模块以处理新的文件格式，创建处理器模块以反汇编新的 CPU 指令集，或者编写用户的其他功能的插件。除此之外，人们为 IDA 开发了大量的有用的插件来扩充其功能，如 IDA Python 插件，它在 IDA 中集成了一个 Python 解释器，可以编写 Python 脚本，实现 IDC 脚本语言的所有功能或者更强大的功能；IDARub 插件在 IDA 中植入一个 Ruby 解释器，并为 SDK 提供 Ruby 接口；以及 IDA Sync、coolabREate、ida-x86emu、mIDA 等功能强大的插件。这一切为逆向技术的发展提供了很大的支持和帮助。

本文主要是在 IDA Pro 反汇编后，使用 Visual Studio 2010 和 SDK 开发工具实现了提取操作码序列的插件。首先使用 IDA Pro 对于样本进行反汇编分析，对于 IDA 的分析结果仔细研究。IDA 的分析结果示例如下图所示，其中图 3-1 为反汇编后的文本视图的一部分，文本视图会显示程序完整的反汇编代码清单，会显示每一条指令的虚拟地址以及一些引用与跳转。文本视图显示程序的信息很完整，方便人们的研究。在文献[35]中就是提取的基于文本的操作码序列作为样本的特征。基于文本的特征提取方法，就是按照反汇编后文本的内容，从文本的开始顺序提取每一条指令的操作码。这样的特征有很大的局限性，对于

代码重排序，垃圾代码的插入，指令的替换等等混淆技术都没有很好的健壮性。本文根据 IDA 提供的便利的接口 startEA，endEA，next_visea()，ua_mnem(),获得对应文本视图的操作码序列以便以后做对比实验。

```
cseg01:0000                 public start
cseg01:0000 start           proc far
cseg01:0000                 push    di
cseg01:0001                 jcxz    short loc_D
cseg01:0003                 push    0
cseg01:0005                 push    0
cseg01:0007                 push    cx
cseg01:0008                 call    LOCALINIT
cseg01:000D
cseg01:000D loc_D:                                  ; CODE XREF: start+1↑j
cseg01:000D                 pop     di
cseg01:000E                 mov     ax, 1
cseg01:0011                 push    ax
cseg01:0012                 push    di
cseg01:0013                 push    si
cseg01:0014                 xor     si, si
cseg01:0016                 xor     di, di
cseg01:0018
cseg01:0018 loc_18:                                 ; CODE XREF: start+29↓j
cseg01:0018                 push    si
cseg01:0019                 call    far ptr GETSTOCKOBJECT
cseg01:001E                 mov     [di+22h], ax
cseg01:0022                 add     di, 2
cseg01:0025                 inc     si
cseg01:0026                 cmp     si, 10h
cseg01:0029                 jbe     short loc_18
cseg01:002B                 pop     si
cseg01:002C                 pop     di
cseg01:002D                 pop     ax
cseg01:002E                 retf
cseg01:002E start           endp
```

图 3-1 IDA 反汇编后的文本视图

图 3-2 为反汇编后的图形视图，从图中可以看出，每一个程序的执行是从 start 函数开始，然后按照程序的控制流程图执行。图形视图中的每一部分叫基本块，清楚地显示了基本块之间的关系，为了方便后面的介绍，本文将基本块简化成为数字 1,2,3,4,5，如图 3-2 中所示。从图形视图中可以很清楚的看到程序的执行路径，即程序的行为。如果在分析程序的特征时能考虑到程序的行为特征，就会更好的描述一个程序。本文结合程序的控制流程图来提取操作码序列，是通过汇编指令的不同功能对反汇编文本构建程序的控制流程图，从程序的 start 函数开始按照程序的执行路径提取每条指令的操作码，并根据控制流程图的分支，提取每一条分支的操作码序列。动态的提取操作码序列，只能提取程序运行时的某一个分支的序列，它不能代表完整的程序。然而，根据控制流程图静态的提取操作码序列，一方面可以按照程序执行的序列提取，这样能很好的描述其行为特征；另一方面考虑了程序各个分支的情况，弥补了动态检测分析不全面的不足。这样的特征不仅仅能很好的诠释样本，更能很好的排除混淆技术给分类带来的干扰。

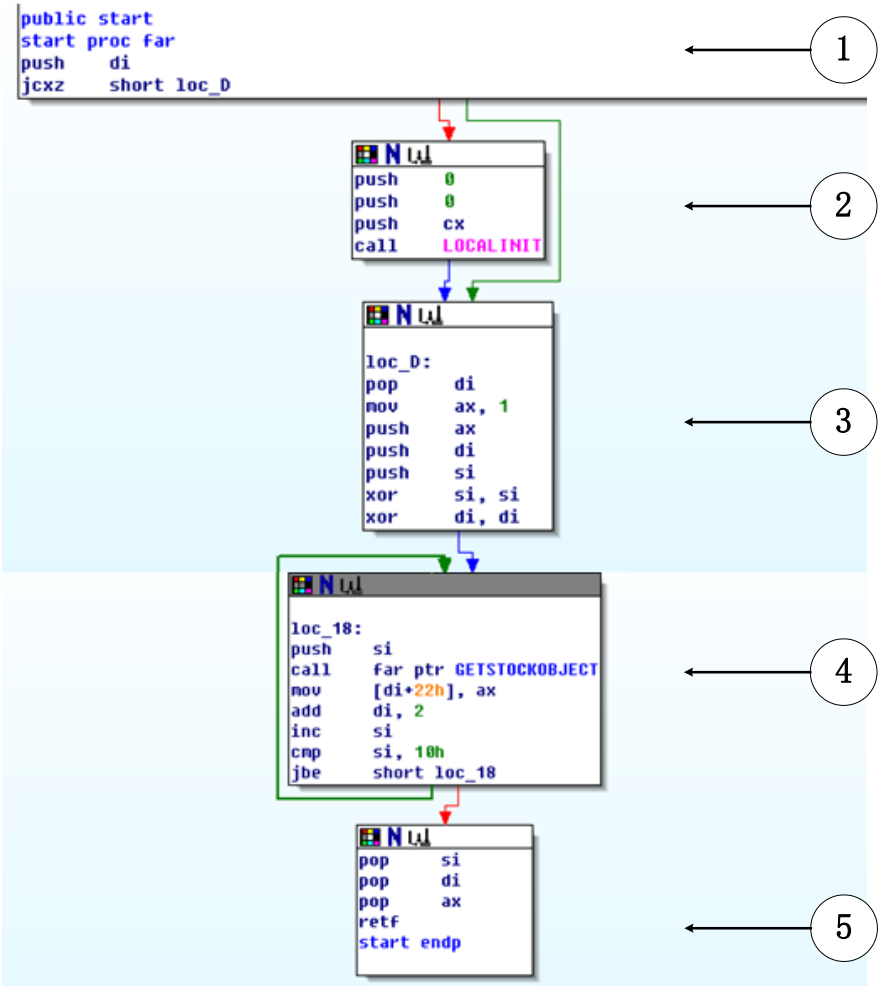


图 3-2 IDA 反汇编后的图形视图

3.3 提取操作码序列

3.3.1 构建控制流程图

IDA Pro 反汇编后的图形视图，即为程序的控制流程图，通过 IDA 插件的编写来构造相似的控制流程图，然后通过深度遍历来提取每一条路径上的操作码序列。

对于汇编代码的分析，可以从数据流、控制流以及程序依赖等方面分析。数据流的分析是对程序中的数据信息，如操作数及变量等信息的关系、定义以及依赖关系的分析。数据分析相对比较复杂，可以分析一些常量，或者结合程序依赖关系，把操作码和操作数结合起来分析。本文主要分析程序的控制流和操作码之间的依赖关系，根据每条指令的关系构造合理的控制流程图。每一条汇编指令都当成一个节点。图中的节点的数据结构定义为：

```
Class Node
{
   public:
      ea_t address;
      string type;
      int child_num;
      Node**child;
};
```

其中 address 是指令的地址，type 是指令的类型，child_num 是指令后继节点的个数，child 是指向后继节点的指针数组。

指令类型为顺序指令、条件分支指令、无条件分支指令、函数调用指令、返回指令。按照指令的类型来定义它的后继节点。顺序指令的后继节点为下一条地址的指令；条件分支有两个后继节点，分别代表两条路径；无条件分支指令的后继节点即为跳转的地址的指令；函数调用指令的后继节点即为函数的起始地址（函数调用本文分为两种，一种是调用程序中的函数，如 call sub_406080，另一种认为是调用系统函数按顺序指令处理）；返回指令的后继地址即为栈顶指令的地址。构造控制流图的算法如表 3-2 所示。

（1）首先 GetStartFunction()函数应用 SDK 接口 get_func_name()接口找到 start 函数的首地址，作为整个流程图的开始节点，其次使用 Initialize()初始化节点的属性并标记该节点被访问过,其类型标记为顺序指令，然后入栈。

（2）出栈，GetSuccessorNode()按照节点的类型找到后继节点，如果后继节点没被访问则把地址赋值给指向后继的指针 child。并将后继节点入栈并标记为该节点的后继节点已经处理过，否则遇到回路会继续处理它的后继节点造成死循环。

（3）如果栈不为空，则执行(2)，否则返回开始节点。

表 3-2 构建控制流程图的算法描述

算法 1： 构建程序的控制流程图

Input: disassembled code

Output: the start node of the control flow graph

```
Start=GetStartFunction();
Initialize(Start);
Stack.Push(Start);
```

表 3-2（续表）

算法 1： 构建程序的控制流程图

```
While (NotEmpty(Stack)){
        CurrentNode=Stack.pop();
        Suc=GetSuccessorNode(CurrentNode);
        If (IsNotVisited(Suc)){
        Stack.push(Suc);
        Visited(Suc);
        CurrentNode->child=Suc;
        }
  }
  Return Start
```

为了简化图形，将图 3-2 模块化表示，根据控制流程图的构建算法可以得到图 3-2 所示的控制流程图，如图 3-3 所示。

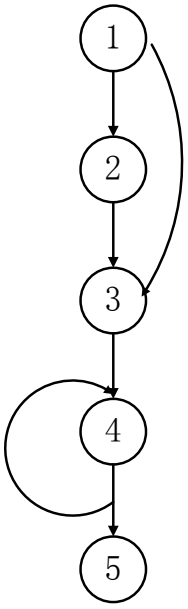


图 3-3 对图 3-2 所示代码构建的控制流程图的简化

在构造的过程中有一个特殊问题需要处理，如图 3-4 所示

```
.text:01002348 ; [0000000A BYTES: COLLAPSED FUNCTION start. PRESS KEYPAD "+" TO EXPAND]
.text:01002352                 db 13h dup(0CCh)
.text:01002365
.text:01002365 ; =============== S U B R O U T I N E =======================================
.text:01002365
.text:01002365
.text:01002365 sub_1002365     proc near               ; CODE XREF: __local_unwind4+AB↓p
.text:01002365                                         ; sub_1006042+A3↓p ...
.text:01002365                 cmp     ecx, dword_1008010
.text:0100236B                 jnz     short loc_1002370
.text:0100236D                 retn    0
```

图 3-4 反汇编后 start 函数被压缩

```
.text:01002348
.text:01002348 ; ================ S U B R O U T I N E ============
.text:01002348
.text:01002348 ; Attributes: library function
.text:01002348
.text:01002348                 public start
.text:01002348 start           proc near
.text:01002348                 call    sub_1002931
.text:0100234D                 jmp     ___tmainCRTStartup
.text:0100234D start           endp
.text:0100234D
.text:0100234D ; ---------------------------------------------------
```

图 3-5 start 函数展开后的结果

有一些样本 IDA Pro 分析后的汇编代码的 start 函数被压缩，因为 start 函数是开始节点认为是顺序指令，在压缩状态下，用 SDK 接口的 next_visea()函数就会执行下一条指令。如图 3-4 所示，下一条指令是 01002365，这样提取的序列是很不理想的，不是按照控制流提取的。所以本文展开 start 函数，如图 3-5 所示，这样下一条指令执行 0100234D，是控制流图的执行顺序。本文中构建控制图时，对于一些 start 函数被压缩的样本，使用 Unhide 插件功能做了展开的处理。

3.3.2 提取操作码序列

控制流程图是一个有向图，图中可能会包含一些回路。为了得到程序的所有可能的执行路径，要消除图中的回路。为了简单，本文对于回路只做一次处理。本文构造一颗没有回路的树。构造树的算法如表 3-3 所示。

（1）使用 CreateRootNode()创建一个树节点，把 start 开始节点当做根节点压入栈中，并将控制流程图的节点都初始化成未标记。

（2）当栈不为空时，出栈并使用 GetExpandableNodes()对栈顶节点扩展，如果其后继节点与当前路径不构成回路为可扩展节点，则将该后继节点入栈并做标记，如果构成回路，则看是否处理过，保证对每一个回路只处理一次。InsertChildrenNode()将可扩展的节点插入作为栈顶节点的孩子节点，并使用

MoveVisitedNode()将已经扩展的节点从原来的未被访问过的集合中删除。

（3）如果栈顶节点没有可扩展的节点，则为一条路径，然后返回到最近的分支，处理其他路径。

（4）如果所有节点都被处理过，则返回树的根节点。

表 3-3 构建控制流程图的执行树的算法描述

算法 2： 构建控制流程图的执行树

```
Input: the CFG
Output: the root node of the running tree
    Root=CreateRootNode(CFG);
    Initialize(CFG);
    Stack.Push(Root);
    While (NotEmpty(Stack)){
        CurrentRoot=Stack.pop();
        ExpandableNodes=GetExpandableNodes(CurrentRoot.UnvisitedChildrenSet);
        If (IsNotNull(ExpandableNodes)){
        ExpandableRootNode=CreateRootNode(CFG);
        InsertChildrenNode(CurrentRoot, ExpandableRootNode);
        ExpandableRootNode.UnvisitedChildrenSet=All children of  ExpandableNodes
        Stack.push(ExpandableRootNode);
        MoveVisitedNode(CurrentRoot.UnvisitedChildrenSet, ExpandableNodes);
        }
        Else
        Stack.pop();
    }
    Return Root
```

经过消除回路，则控制流程图 3-3 被转化成一棵二叉树，如图 3-6 所示。

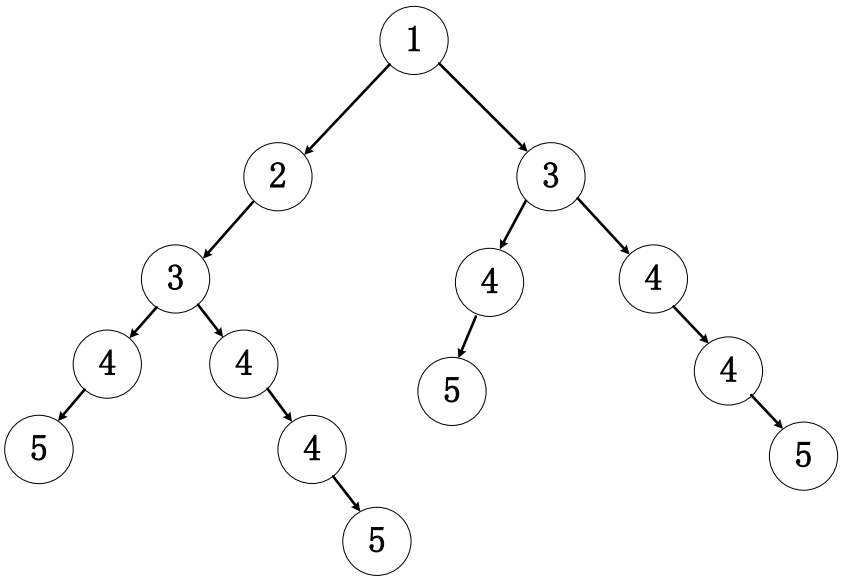


图 3-6 将控制流程图转化为二叉树的结果

让一个循环执行一次，这样就会消除图中的回路。如果一条执行路径中，一个节点出现两次，则存在回路。从树的根节点到每一个叶节点即为一条执行的路径。理论上，可以获取程序的每一种可能的执行路径。通过遍历树的每一条路径，就会得到所有的操作码序列。

从图 3-7 可以看出，每一条操作码序列都是以 retf 结束（也有的程序是以 retn 结束，其中 call 指令不是 call sub 类型的按顺序指令处理）。根据结果可以分析得出，其执行的路径与图 3-2 所示的结果一致。

```
push jcxz pop mov push push push xor xor push call mov add
     inc cmp jbe push call mov add inc cmp jbe pop pop pop retf
push jcxz pop mov push push push xor xor push call mov and inc cmp
     jbe pop pop pop retf
push jcxz push push push call pop mov push push push xor xor push
     call mov add inc cmp jbe push call mov add inc cmp
     jbe pop pop pop retf
push jcxz push push push call pop mov push push push xor xor push
     call mov add inc jbe pop pop pop retf
```

图 3-7 基于控制流程图提取的操作码序列的结果

基于文本的操作码序列的提取方法，是根据反汇编文本的内容，按照顺序依次提取操作码序列，就上例而言，基于文本的操作码序列的提取结果为：push, jcxz, push, push, push, call, pop, mov, push, push, push, xor, xor, push, call, mov, add, inc, cmp, jbc, pop, pop, pop, retf。

3.4 操作码序列特征的分析

3.4.1 操作码特征的比较

通过统计基于文本和控制流程图两种方法得到的操作码的种类发现，基于文本的操作提取方法一共有 284 种操作码，而基于程序的控制流程图的方法只有 151 种。这是因为基于控制流程图的方法是按照程序执行时的路径来提取，这样提取的操作码只是整个汇编文本的一部分，大都是一些汇编指令，而基于文本的方法是根据内容依次提取，所以基于文本的方法提取的操作码种类会比较多。从操作码种类的结果发现，基于文本的方法提取了很多 SSE2（Streaming SIMD Extensions）数据流单指令多数据扩展指令集中的指令，如数据移动指令 movapd、算数指令 addpd、比较指令 cmppd 和数据重组和解包指令 unpckhpd 等等。此外，本文有一些样本是基于.NET 平台开发的，反汇编后文本使用 IL（中间语言）表示，所以提取了很多 IL 指令如 ble.un.s、conv.i1、ldc.i4.2 等。因为这种样本很少得到的操作码没有实际的分类意义，所以本文没有使用这种样本。反汇编后用中间语言表示的情况如图 3-8 所示。

```
loc_5CE7:                               // CODE XREF: Main+1E↑j
  .try {
    ldloca.s 0
    ldfld class System.String Microsoft.VisualStudio.TestTools.UnitTesting.Publicize.CommandLineSwitches::AssemblyName
    call class System.String [mscorlib]System.IO.Path::GetDirectoryName(class System.String)
    stloc.1
    leave.s loc_5D36
  }
  catch [mscorlib]System.ArgumentException {
    stloc.2
    call class System.String Microsoft.VisualStudio.TestTools.UnitTesting.Publicize.Properties.Resources::get_AssemblyLoadError()
    ldloca.s 0
    ldfld class System.String Microsoft.VisualStudio.TestTools.UnitTesting.Publicize.CommandLineSwitches::AssemblyName
    ldloc.2
    callvirt class System.String [mscorlib]System.Exception::get_Message()
    call void [mscorlib]System.Console::WriteLine(class System.String, class System.Object, class System.Object)
    ldc.i4.2
    stloc.s 7
    leave loc_5DC7
  }
```

图 3-8 反汇编后使用中间语言表示的示例

3.4.2 操作码序列的分析

恶意代码的执行往往不同于正常的程序，它的操作码序列也有相应的特征，例如一些恶意代码为了躲避检测，会使用大量的连续的 jmp 指令以达到最终跳转到真正的指令的目的；也有一些代码会使用大量 call 指令调用和跳转指令来混淆其真实行为；还有一些通过垃圾代码的插入和变量的重命名，这使得汇编指令中有很多连续的 push 指令；还有一些通过使用大量 mov 和 lea 对数据的转换实现恶意代码的变种；也有一些恶意代码使用大量的 nop 指令来延时等等。

如图 3-9 所示，这是一个后门程序控制流程图的一个分支的操作码序列，通过连续使用 53 个 jmp 指令来隐藏其真实的行为。

```
push call push push push push push call push call int jmp jmp jmp jmp
jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp
jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp
jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp jmp icebp mov jb mov mov
mov dec icebp mov retf
```

图 3-9 后门程序 Backdoor.Win32.Angent.mt 的一个分支的操作码序列

分析通过控制流程图提取的操作码序列并统计分析特征文件的属性：得到特征序列文件的大小和源文件的大小的比值。如表 3-4 所示 Low 表示源文件与特征序列文件比值小于 1 的平均值，High 表示源文件与特征序列文件比值大于 1 的平均值。可以发现，恶意代码的比值要远远大于或远远小于正常样本的比值。这是因为有一些恶意代码为了躲避检测，使用了代码混淆技术：垃圾代码插入技术。这样会使得，程序中有很多垃圾代码，而程序不会运行这些代码，程序的控制流程图非常简单，然而本文根据程序的控制流程图提取特征，这就使得提取的特征的文件与源文件的比值非常小。相反，有的程序通过不断地跳转，或不断的调用已有的函数实现恶意的行为，使得原程序很小但是程序的控制流程图非常复杂，所以在提取特征时会产生很多操作码序列，这就使得提取的特征文件与源文件的比值会很大。如果根据文本提取操作码序列，这样提取的文件的大小与源程序的大小是成正比的。

表 3-4 所有样本的特征序列文件和源文件的大小比值的平均值

	正常样本	恶意样本
Low	0.368043	0.09864
High	97.8883	1037.267

3.4.3 n-gram 算法提取特征

n-gram 模型被广泛应用在自然语言处理，语音识别等领域。该模型基于第 n 个词出现只与前 n-1 个词相关的假设，对于处理依靠序列关系分类的问题有很强的优势。本文应用 n-gram[45] 的思想是：将获得的操作码序列，根据大小为 n 的滑动窗口，获得一系列的长度为 n 的序列，具体表示如下：

```
pop     di
mov     ax, 1
push    ax
push    di
push    si
xor     si, si
xor     di, di
```

图 3-10 一段反汇编代码示例

定义一个程序为 p，操作码序列为 o，则可表示为 $p=(o_1,o_{2,}\cdots o_l)$，其中 l 表示一个程序操作码序列的长度。这样每一个特征都是它的子序列。对图 3-10 中给出的汇编代码应用 3-gram 算法可获得特征为：$s_1=(pop,mov,push)$，$s_2=(mov,push,push)$，$s_3=(push,push,push)$，$s_4=(push,push,xor)$，$s_5=(push,xor,xor)$。为了实现简单，把每一个操作码对应成一个整数。对于 n-gram 算法，很难确定滑动窗口的长度，太小很难检测复杂的模块，太大又很难检测一些简单的混淆技术。本文通过实验对比，选择了特定的长度，n 分别选择 3、4 及 5 来提取相应的特征。

3.4.4 信息增益

首先介绍熵的概念，熵是香农在 1984 年提出的解决信息论领域问题的重要概念。一个随机变量的熵越大，其所含的信息量就会越大。熵的计算公式如下：

$$H(X)=-\sum_{x\in X}p(x)\log_2 p(x) \tag{3-1}$$

在随机变量 Y 发生的情况下，X 的信息熵为：

$$H(X/Y)=-\sum_j p(y_i)\sum_i p(x_i\mid y_j)\log_2(p(x_i\mid y_j)) \tag{3-2}$$

信息增益为信息熵的差：

$$IG(X,Y)=H(X)-H(X/Y) \tag{3-3}$$

信息增益是一个非常重要的概念，通过计算信息增益的方法来选择特征被广泛应用。在本文中，计算每一个特征序列 s_i 的信息增益，由于对于固定的数据集，类别的熵值是固定的，所以可以简化计算信息增益的公式为：

$$IG(s_i)=\sum_{v_i\notin\{0,1\}}\sum_{c_j\in\{C\}}p(v_i,c_j)\log_2\frac{p(v_i,c_j)}{p(v_i)p(c_j)} \tag{3-4}$$

其中，s_i 代表一个特征序列，当 s_i 在训练样本中出现则 v_i 的值为 1，否则为

0；C为类别，本文中只有 0 或 1 两种情况，0 表示恶意样本，1 表示正常样本。$p(v_i,c_j)$表示在类别c_j中s_i出现或者不出现的比例，$p(v_i)$表示样本中特征s_i出现或者不出现的样本比例，$p(c_j)$表示整个实验数据中类别c_j的比例。

信息增益越大，则对应的特征的区分能力越好。因此，本文选择信息增益大的值，在选择特征时设了一个阈值，只有大于这个阈值的信息增益的特征才会被选择，因为信息增益小的特征，区分度比较小，不仅不能起到很好的分类效果，反而有可能会降低分类的效果。

如表 3-5 所示，表中列举了当特征长度 n=4 时，两种特征提取方法的特征的信息增益分布情况。基于控制流程图的特征提取方法中信息增益大于 1.8 的特征数占总特征数的比例约 1.6%，基于文本的特征提取方法的特征数量中信息增益大于 1.8 的比例约为 0.07%。一般信息增益大于 1.8 的特征有很好的区分能力，而基于控制流程图的方法中信息增益大于 1.8 的特征比例较大，这说明基于控制流程图的方法能提取出更有区分能力的特征。

表 3-5 特征长度 n=4 时所有特征的信息增益分布情况

	IG < 1.4	1.4≤IG < 1.8	IG≥1.8
基于控制流程图提取的特征	0.866529	0.117563	0.0157493
基于文本提取的特征	0.782203	0.217082	0.00070826

3.4.5 文档频率

文档频率，使用特征在某一类别中出现的次数表示该特征与类别的相关度。用(3-5)式来表示特征每类中出现的次数与其相关度。

$$DF(s,c)=\frac{\lg(DF_s)}{\lg(N_c)} \tag{3-5}$$

其中，DF_s为特征序列s在类别c中出现的文档频率，N_c为类c中总的样本数。同样也设置一个阈值，如果计算得到的值小于阈值，则该特征序列在对应的类别的样本中出现的次数太少，没有很好的代表性，对于正确的分类不仅仅没有很好的贡献，反而会使精度下降。还有一些特征序列的相关度值在两种类别中都很高，这样的特征序列对分类的贡献也不大，会影响精度。选择那些相关度值高并且在两类中相关度值差距比较大的特征作为分类的特征。

文档频率方法与信息增益方法比较起来相对简单，算法复杂度也低。也能

克服信息增益在选择特征时候的偏差。因为信息增益在选择时只根据信息增益的大小来选择，这样有可能选择的特征会在某一类别中出现的次数比较多而在另一类别出现的次数比较少。为了更好的解决这一不平衡问题，在选择特征的时候尽量按比例分别从两类样本中选择特征。

图 3-11 为特征长度 n=4 时，从恶意样本中提取出前 50 个文档频率高的特征并从正常样本中也提取出前 50 个文档频率高的特征的分布图。其中，左图为基于控制流程图特征提取方法的文档频率分布，右图为基于文本的特征提取方法的文档频率分布。左图中，公共的特征有 9 个并且大部分特征在一类中出现次数多而在另一类出现次数少，而在右图中，公共特征有 18 个且大部分特征在两类中出现的次数相差不多。对比图中文档频率的分布，可以得出基于控制流程图的方法提取的特征有很好的区分能力，而基于文本的方法提取的特征在两类样本均多次出现没有很好的区分能力。

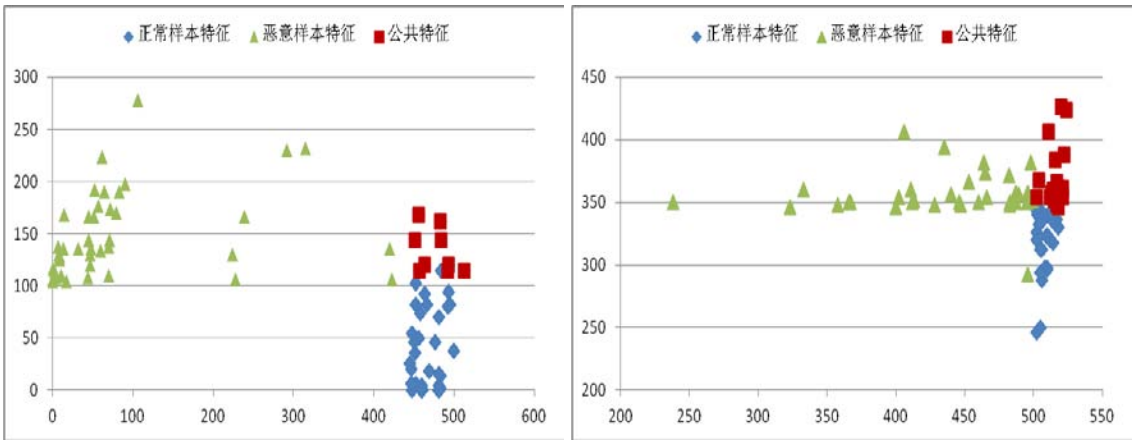


图 3-11 文档频率的对比图

3.5 恶意代码的分类

3.5.1 决策树

决策树（Decision Tree），是通过归纳学习形成的一种树结构。一颗决策树由根节点，内部节点和叶子节点组成。其中，每一个内部节点代表决策过程中要测试的属性，叶节点代表分类的结果[46]。

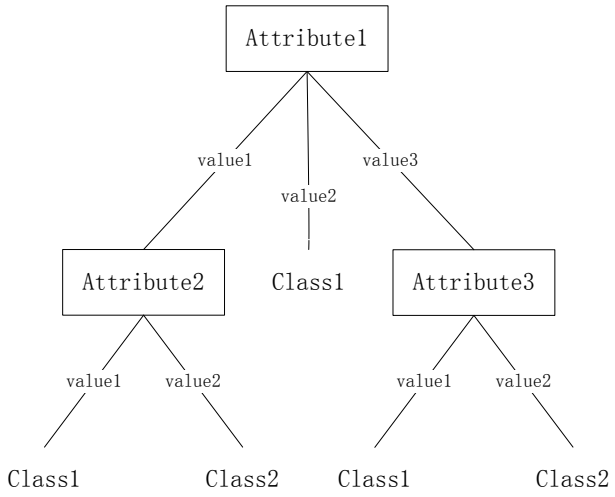


图 3-12 决策树模型

80 年代初期，Quinlan 开发了决策树算法 ID3，后来又提出 C4.5 成为新的监督学习算法的性能比较基准，接着有人提出分类与回归树 CART。它们都采用贪心（非回溯）方法，以自顶向下递归的分治方式构造决策树模型。ID3 算法的基本思想：采用递归的方法，首先找出区分能力强的属性，把样本分成多个子集，每个子集又选择区分能力强的属性进行划分，直到所有的子集只包含同一类别的数据为止，叶节点标记为该类别。

ID3 使用信息增益作为属性选择度量，这样会偏向于选择属性取值较多的属性，然而这不一定是最好的分类属性。所以本文采用改进后的 C4.5 算法。该算法增加了属性值空缺情况的处理，能够对连续属性离散化处理。采用悲观剪枝技术，剪去不能提高预测准确率的分支，并先剪枝和后剪枝交叉使用以避免过度拟合问题。C4.5 算法为了避免 ID3 选择属性时产生的偏倚，采用信息增益率来选择属性，选择具有最大的增益率作为分裂属性。信息增益率的计算方法如下：

$$GainRatio(A)=\frac{Gain(A)}{SplitInfo(A)} \tag{3-6}$$

其中，$SplitInfo(D)=-\sum_{j=1}^{v}\frac{|D_j|}{|D|}\times\log_2(\frac{|D_j|}{|D|})$ (3-7)

其中 D 为数据集，A 表示属性，将 D 划分为 V 个子集$\{D_1,D_2,\ldots D_v\}$，$Gain(A)$为属性 A 的信息增益。

3.5.2 KNN

KNN（K-Nearest Neighbor）算法的主要思想是：通过对数据预处理，得到每一个样本用一个向量表示，向量的最后一维代表样本的类别。采用一种度量的方法计算样本之间的相似度，计算出与被测样本最近的 K 个样本（K 一般选择奇数），然后统计这 K 个样本中每个类别出现的次数，被测样本的类别与出现次数最多的类别相同。如图 3-13 所示，在实线圆圈内 K=3，测试样本圆点被分类为三角形，而在虚线圆圈内 K=5，测试样本被分类为正方形。

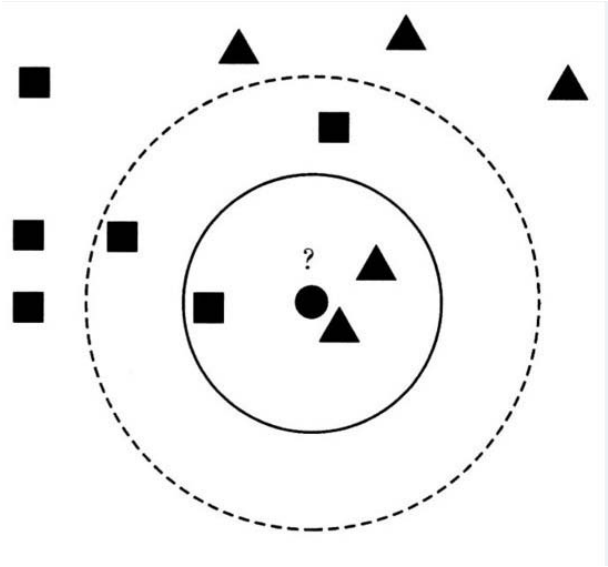


图 3-13 KNN 模型分类的示意图

在计算前，首先要对特征向量进行归一化，本文采用的是线性函数的转换方法：

$$y = (x - Min) / (Max - Min) \tag{3-8}$$

其中，x, y 分别为转换前、后的值，Max, Min 分别为样本的最大值和最小值。

设归一化后的特征向量为 $v = (s_1, s_2, \ldots s_l)$，则两个向量 v_1, v_2 的相似度可以有很多表示方式，本文使用欧氏距离：

$$d(v_1, v_2) = \sqrt{\sum_{i=1}^{l} (v_1(s_i) - v_2(s_i))^2} \tag{3-9}$$

选择距离最近的 K 个样本，则目标函数为：

$$f(v)=\arg\max_{c\in C}\sum_{i=1}^{k}\delta(c,f(v_i)) \tag{3-10}$$

其中，$f(v)$表示特征向量 v 的类别，如果 $f(v_i)$的类别值与 c 类别相同则 $\delta(c,f(v_i))=1$，否则 $\delta(c,f(v_i))=0$。

K 近邻是一种简单的算法，被广泛应用于文本分类，它对于 K 的值依赖很大，K 一般选择奇数。没有一个标准的方法来选择 K 的值，本文通过实验选择分类效果较好的 K 值。

3.5.3 SVM

SVM（Support Vector Machine）支持向量机，是一种基于统计学理论发展起来的机器学习分类方法。它被广泛应用于手写识别、文本分类、图像分类等。SVM 适合解决高维及非线性等问题。SVM 算法的基本思想：通过计算寻找最优的线性超平面，如果问题是非线性可分的，通过某种非线性映射把样本空间映射到一个高维的特征空间，在新的特征空间中搜索最优的线性超平面。

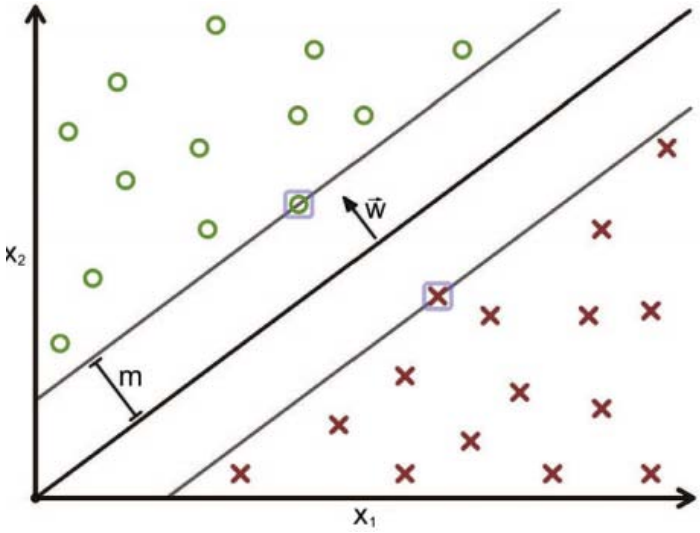


图 3-14 在平面上 SVM 分类的示意图

SVM 在空间里建立最大间隔超平面，使得超平面与两类分别最近的距离尽量相等。如果线性可分，在超平面在两类样本间距离最小的两个样本连线的中垂线上。SVM 是典型的两类分类器，很适合两类的文本分类的研究问题。SVM 的关键在于核函数，常用的核函数有四种：

（1） 线性核函数 $K(x,y)=x\cdot y$；

（2） 多项式核函数 $K(x,y)=[\gamma(x\cdot y)+r]^d,\gamma>0$；

（3） 径向基函数 $K(x,y)=\exp(-\gamma\|x-y\|^2),\gamma>0$；

（4） 二层神经网络核函数 $K(x,y)=\tanh(\gamma(x\cdot y)+r)$；

其中 γ,r 和 d 都是核参数。

核函数的作用将两个低维空间里的向量变换为高维空间里的向量内积值。核函数的选择也没有固定的规则，不一样的问题选择的核函数不同，本文通过实验比较选择多项式核函数。

3.6 本章小结

本章主要介绍了基于操作码序列的恶意代码检测方法。重点介绍了提取操作码序列的步骤：首先，对数据查壳和脱壳处理并用 IDA Pro 对样本进行反汇编分析，提取基于文本的操作码序列 ；其次，构建控制流程图并且消除图中的回路；然后构造树并遍历树的每一条执行路径得到所有可能的操作码序列。最后把所获的结果与 IDA Pro 分析的控制流程图比较，结果一致。接下来介绍了使用 n-gram 算法提取特征的过程及信息增益和文档频率选择特征的方法。最后，简单介绍了三种机器学习分类算法，决策树、K 近邻及支持向量机。

第 4 章 系统设计与实验结果

4.1 系统设计

下面详细介绍本文的整个系统框架。首先，将恶意代码进行查壳与脱壳处理；其次，利用 IDA Pro 反汇编工具获得基于控制流图和基于文本的操作码序列；再次，使用 n-gram 算法提取特征，然后采用信息增益和文档频率选择区分度高的特征；最后，使用机器学习分类算法实现分类，并对比实验结果。系统框架如图 4-1 所示。

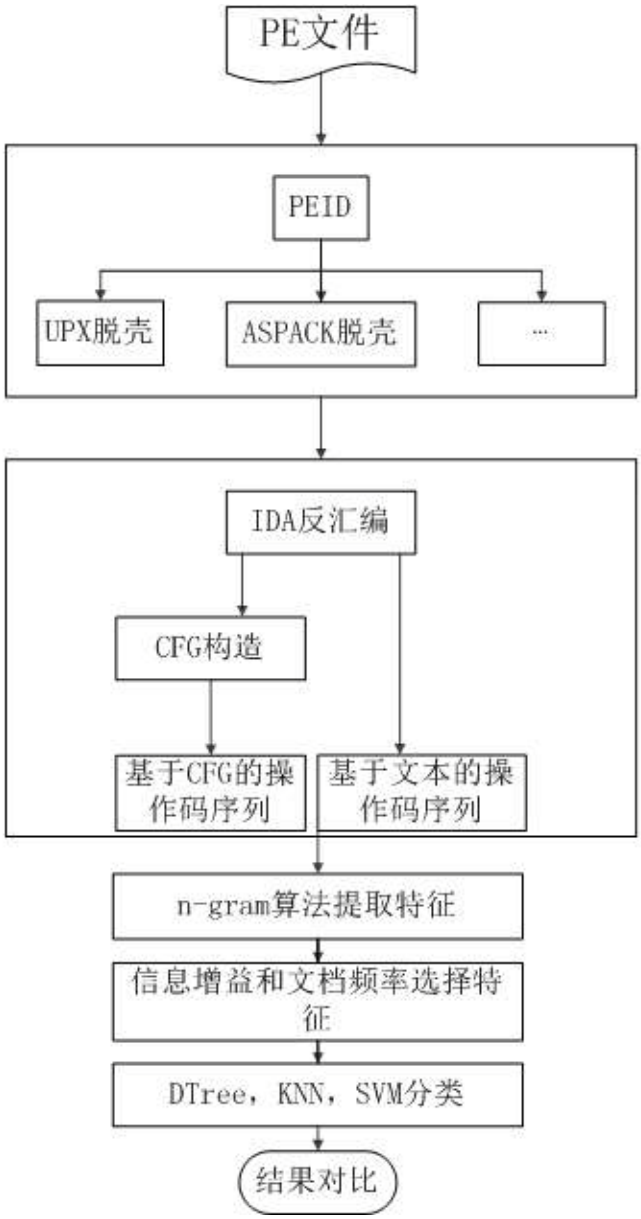


图 4-1 系统的框架

本文的系统在 Windows7 系统中的 Visual Studio 2010 平台下，使用 IDA Pro 5.2 版的 SDK 和 C++语言开发完成。首先，使用 PEID 和 VMUNPACKER 对恶意代码进行查壳和脱壳处理，其中有压缩壳，如 UPX、FSG 等，也有加密壳，如 nSpack、ASPack 等，壳的种类有很多，但是大部分恶意代码加的壳是 UPX、ASPack、FSG 及 PECompact。其次，使用 IDA Pro 对样本进行反汇编分析得到汇编代码文本，并根据 SDK 提供的接口按照文本的内容顺序提取操作码的序列。再次，将每一条指令看做一个节点，根据操作码的不同类型确定其后继节点的地址。利用 SDK 接口找到程序的入口函数 start 函数，并按照节点的类型构造程序的控制流程图。为了得到每一条执行路径的操作码序列，本文对流程图中每一个回路处理一次来消除回路，最终构建程序的执行树。从根节点到每一个叶节点即为一条路径，使用深度优先遍历算法来提取每一条路径的操作码序列。然后，考虑到序列之间的关系使用 n-gram 模型提取操作码序列作为特征，并计算每一个特征的信息增益和文档频率，选择信息增益大的及文档频率高且在两类样本中频率差距较大的特征作为分类特征。最后，使用 K 近邻、决策树及支持向量机实现了恶意代码的检测。

4.2 评价方法

K 重交叉验证（K-Fold Cross Validation）在人工智能、模式识别、机器学习及分类器研究等方面有很强的指导和验证意义。主要思想是：把实验数据分为 K 组，将每组分别作为测试集，其余 K-1 组作为训练集，这样会得到 K 个模型，然后将 K 个模型得到的结果的平均数作为分类器的性能指标。这样会避免测试样本的特殊性造成结果的不准确问题。在交叉验证划分数据集时也要注意，尽量均匀取样，把每类样本均分成 K 份，然后分别把每类样本中的其中一份组合起来，这样数据划分会均匀。在本文中，采用的是 5 重交叉验证，并且把恶意样本和正常样本分别分成 5 份，然后再组合起来以达到样本划分均匀的目的，最后计算分类结果的平均值。

对于恶意代码的检测系统，一般采用准确率（Accuracy），误报率（FPR）以及漏报率（FNR）来评价其效果。

用 TP 表示，恶意样本被正确分类的数量；FN 表示恶意样本被分为正常样本的数量；TN 表示正常样本被正确分类的数量；FP 表示正常样本被分为恶意样本的数量。则计算公式如下：

$$Accuracy = \frac{TP + TN}{TP + FP + TP + TN} \tag{4-1}$$

$$FPR = \frac{FP}{FP + TN} \tag{4-2}$$

$$FNR = \frac{FN}{TP + FN} \tag{4-3}$$

4.3 实验数据

本文所使用的实验数据包括 570 个正常样本和 483 个恶意样本，样本均为 PE 格式文件，有病毒、木马、蠕虫及后门等。其中恶意样本来源于 http://vx.netlux.org 和 http://www.offensivecomputing.net 两个网站，正常样本来源于 window 的系统文件及一些其他的正常文件，并通过 360 杀毒软件验证。实验的过程中，使用了 K 重交叉验证，所以正常样本和恶意样本分别分成五份，然后同时从恶意样本和正常样本中取一份作为测试集，其余均为训练集。

4.4 实验结果对比与分析

下面将详细的介绍实验结果的对比和分析。主要从准确率、误报率及漏报率三个方面评价实验效果。通过使用不同的特征长度 n 值以及不同的特征数量对比三种分类算法的实验结果。其中，K 近邻算法中的 K 通过实验对比选择 K 值为 9。同样的方法，支持向量机算法中的核函数本文选择多项式核函数。

4.4.1 准确率比较

下面通过不同的 n 值和不同数量的特征来对比准确率。

图 4-2 中，左图是使用信息增益特征选择方法的结果，右图为使用文档频率特征选择方法的结果，且以后的对比图中均是这样排列。其中，横坐标均为所取特征的数量，纵坐标均为准确率。其中 CFG-DT、CFG-KNN、CFG-SVM 分别表示通过控制流程图提取操作码序列的决策树、K 近邻、支持向量机的分类结果。同理，NO-DT、NO-KNN、NO-SVM 分别表示根据反汇编文本提取操作码序列的决策树、K 近邻、支持向量机的分类结果。

通过对比图 4-2 中结果，当 n=3 时可以得出如下几点结论：

（1）对于三种分类算法，使用信息增益的特征选择方法的分类效果比基于文档频率特征的稍微好一些，平均高出 2%。

（2）从图中的曲线看出特征数量的选择没有很好的规律性，基于信息增益的特征选择方法中最好的准确率接近 95%，在特征数量为 40 时 SVM 算法达到最低的准确率接近 80%，而基于文档频率的特征选择方法中，在特征数为 100 时决策树算法达到最好的分类效果接近 94%，在特征数量为 20 时 SVM 算法达

到最低的准确率约 78%。

（3）不管是哪种特征提取方法，决策树算法的准确率要比其他两个分类算法的效果好，而且决策树算法的分类效果一般平均在 90%到 95%之间。支持向量机的分类算法效果相对较差些，一般平均在 75%到 90%之间。

（4）无论是信息增益还是文档频率选择特征，基于控制流程图的特征提取方法的分类结果一般情况下都要比基于文本的效果好。平均准确率能高出 2%。只有在 KNN 分类算法中，在有些特征数量时，分类效果比较相近。

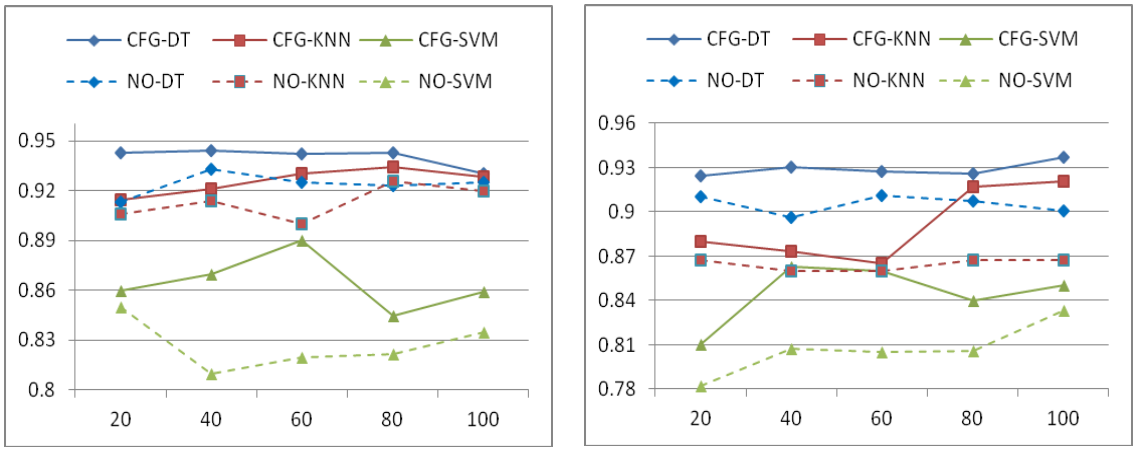


图 4-2 当 n=3 时的准确率对比图

通过对比图 4-3 中结果，当 n=4 时可以得出如下几点结论：

（1）在使用信息增益特征选择方法的分类结果中，特征数量为 60 时分类效果稍微好一些，最高约 95%。在使用文档频率特征选择方法的分类结果中，特征数为 60 的时候效果相对较好，最高接近 94%。

（2）在两种特征选择的方法中，使用三种分类方法的结果由图中可以很明显的看到基于控制流程图的特征提取方法都要比基于文本的效果好，平均高出 2%。

（3）不管是哪种特征提取方法，决策树算法的准确率较高，平均约为 92%。支持向量机的分类算法效果相对较差些，平均约为 85%。

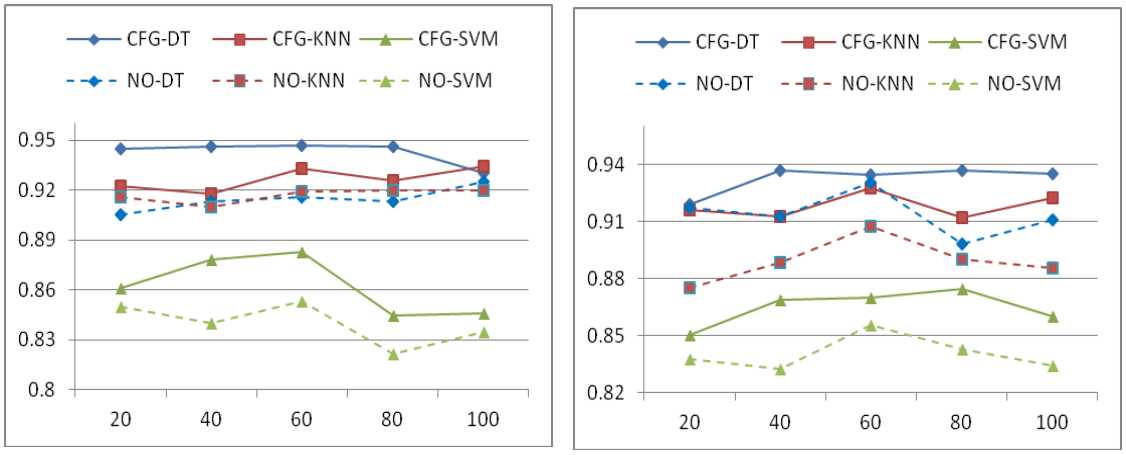


图 4-3 当 n=4 时的准确率对比图

通过对比图 4-4 中结果，当 n=5 时可以得出如下几点结论：

（1）当 n=5 时，在信息增益的特征选择方法中决策树的分类效果没有明显好于 KNN 分类算法，两种分类算法的效果差不多，平均都大于 92%。而 SVM 的分类效果依旧比其他两种分类算法的差，平均约为 85%。

（2）在信息增益的特征选择方法中，基于控制流程图方法的分类效果明显好于基于文本的，大约高 2%。只有在特征数为 100 时，决策树分类算法中基于文本分方法的分类效果稍微好于基于控制流程图的。

（3）在文档频率的特征选择方法中，对于决策树算法，基于控制流程图的特征提取方法的分类效果略好于基于文本的，准确率约为 93%。但是对 KNN 和 SVM 分类算法，基于控制流程图的分类效果要明显好于基于文本的。准确率约相差 3%。

（4）在此次比较中，特征数量的选择对分类效果的影响没有明显的规律。在信息增益特征选择方法中，特征数为 20 和 60 时分类效果稍微好一些，最高约为 95%，而在文档频率的特征选择方法中，没有所有算法均好的特征点，曲线整体比较平稳。

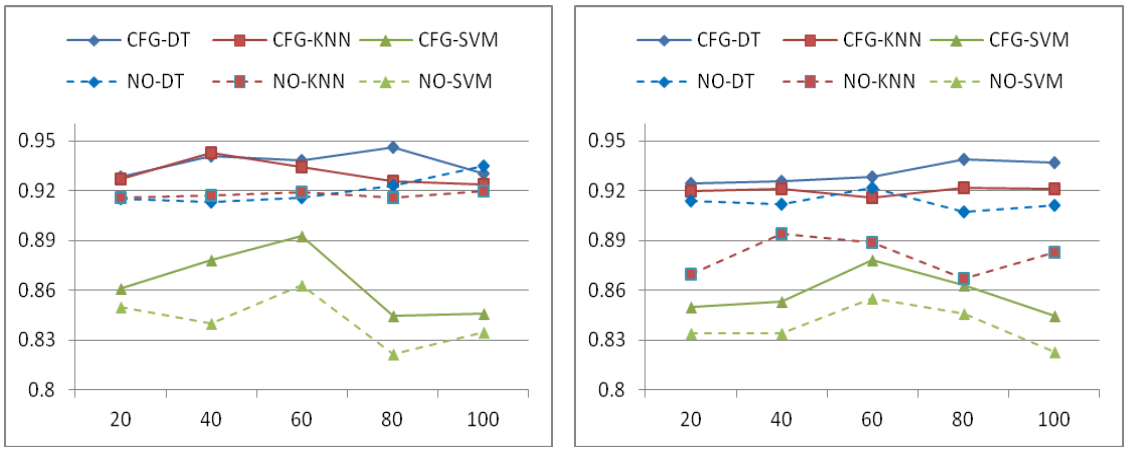


图 4-4 当 n=5 时的准确率对比图

通过上面的比较，可以发现无论哪种特征选择方法，无论哪种分类算法，基于控制流程图的特征提取方法的效果都要比基于文本的效果好一些，一般准确率要高 2%。这种方法的准确率比较高，大约在 90%~96%之间，实验效果比较理想。就两种特征选择方法而言，信息增益的特征选择方法的实验效果比文档频率的好，平均约高 2%。三种分类算法中，决策树的分类效果较好，准确率平均约为 93%，而 SVM 的分类效果较差，约为 85%。通过不同 n 值来使用 n-gram 算法提取特征，从结果的比较来看，n 的选择对准确率有影响，但是无论 n 值选择什么，本文提出的结合控制流程图的基于操作码序列的特征提取方法都要比基于文本的方法好，当 n=4 时，实验对比明显且准确率相对较高，最高约 95%，对于实验中特征数量的选取没有明显的规律。

4.4.2 误报率比较

下面通过不同的 n 值和不同数量的特征来对比误报率。

图 4-5 中，横坐标均为所取特征的数量，纵坐标均为误报率，左图是基于信息增益特征选择方法，右图是基于文档频率特征选择方法。下面的图均如此。当 n=3 时，通过对比观察可以发现：

（1）基于信息增益特征选择方法的误报率比基于文档频率的稍微低一些。在两种特征选择方法中，三种分类算法的误报率都低于 20%。

（2）结合左右两图可看出，无论哪种特征选择方法，决策树算法的误报率均低于其他两种分类算法，其误报率约在 5%到 10%之间。而支持向量机的误报率较高，大约在 12%到 20%。

（3）从图中可以发现，特征数量的选择也几乎没有明显的规律，在特征数为 60 时，所有算法的误报率相对比其他数量的要低一些。

（4）无论哪种提取特征方法，总体上来看对所有分类算法，基于控制流程图的误报率平均低于基于文本的约 2%。只有在基于信息增益的特征选择方法中特征数为 60 时，基于文本的 SVM 算法的误报率低于基于控制流程图的。

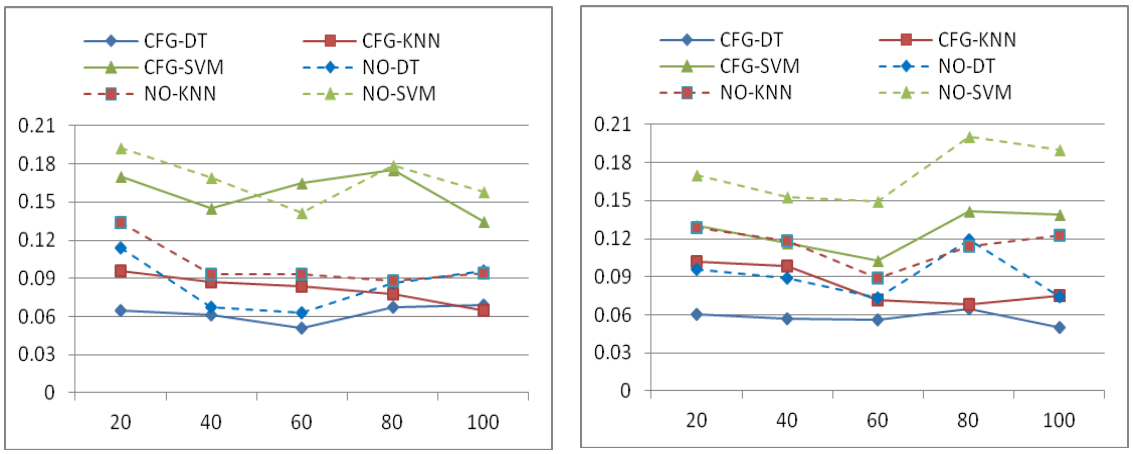


图 4-5 当 n=3 时误报率的对比图

通过对比图 4-6 中结果，当 n=4 时可以得出如下几点结论：

（1）从图中的曲线可以看出，基于控制流程图的特征提取方法的误报率低于基于文本的，只有个别点的结果相反，误报率约在 5%到 20%之间。

（2）两种特征选择方法的结果中，决策树的误报率较低约在 5%到 10%之间，支持向量机的误报率较高约在 10%到 20%之间。

（3）在基于信息增益的特征选择方法中,特征数量的选择没有明显的规律。在基于文档频率特征选择方法中，当特征数量为 60 和 100 时误报率相对较低一些，最低约 5%。

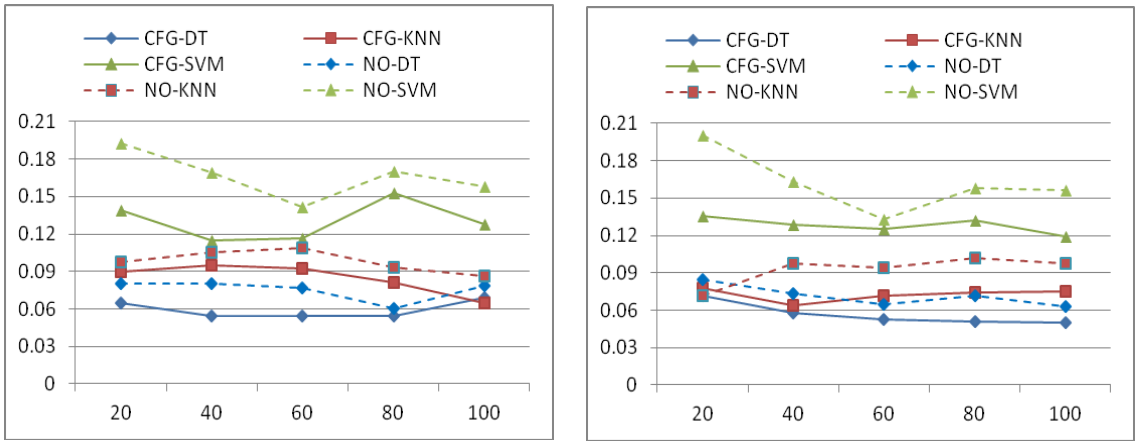


图 4-6 n=4 时误报率的对比图

通过对比图 4-7 中结果，当 n=5 时可以得出如下几点结论：

（1）在基于信息增益的特征选择方法中，总体上基于控制流程图的误报率要低于基于文本的，只有特征数为 80 时，KNN 分类算法中基于控制流程图的误报率略高于基于文本的，误报率约在 5%到 20%的范围内。随着特征数量的增加，支持向量机的误报率先减小后增大，在特征数为 60 时，误报率较低。

（2）在基于文档频率的特征选择方法中，总体上基于控制流程图的误报率也明显低于基于文本的，平均低 2%。只有特征数为 60 时，KNN 算法的结果稍微有偏差，在特征数为 100 时，SVM 算法中基于控制流程图的特征提取方法的误报率较高。

（3）两种特征选择方法的结果中，决策树的误报率较低约在 5%到 10%之间，支持向量机的误报率较高约在 10%到 20%之间。

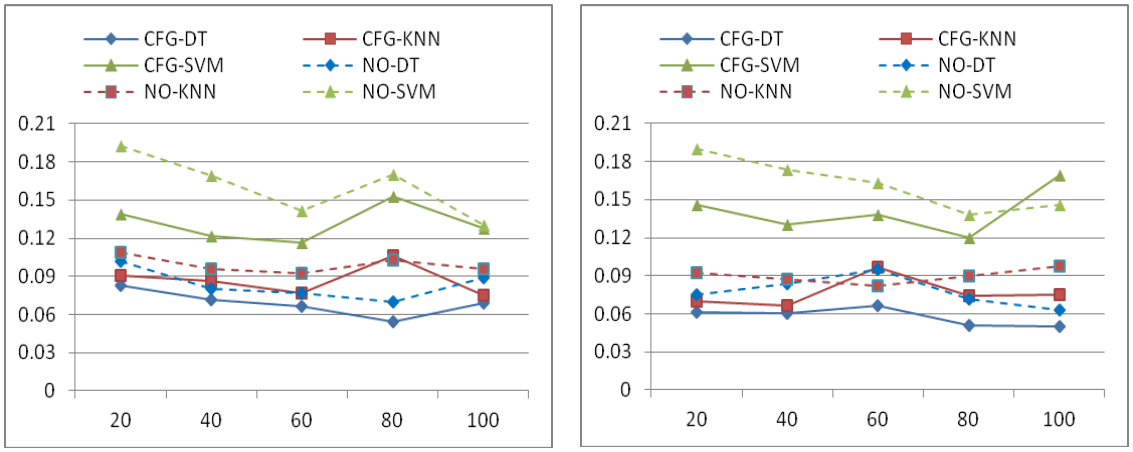


图 4-7 n=5 时误报率对比图

通过对上述三组图的比较和分析可以发现，虽然基于控制流程图的特征提取方法在有些特征数量时的误报率不是一定低于基于文本的，但是总体上基于控制流程图的误报率要相对低一些，平均低 2%。无论哪种特征选择方法，决策树分类算法的误报率较低约为 5%到 10%，支持向量机的误报率较高，约为 10%到 20%。当 n=4 时，实验对比效果明显且误报率相对较低。而特征数量的选取与误报率的关系并不明显。

4.4.3 漏报率比较

下面通过不同的 n 值和不同数量的特征来对比漏报率。

图 4-8 中，横坐标均为所取特征的数量，纵坐标均为漏报率，左图是基于信息增益特征选择方法的，右图是基于文档频率特征选择方法的。下面的图均如此。通过对比图 4-8 中结果，当 n=3 时可以得出如下几点结论：

（1）在基于信息增益的特征选择方法中，基于控制流程图的漏报率要低于

基于文本的；三种分类算法中 KNN 和决策树的漏报率较低，约为 3%到 11%，SVM 算法的漏报率较高约为 5%到 21%。

（2）在基于文档频率的特征选择方法中，基本上是基于控制流程图的漏报率较低，平均低 3%。三种分类算法中决策树的漏报率较低，约为 6%到 12%，SVM 算法的漏报率较高约为 15%到 27%。

（3）通过两图的比较，可以发现，基于文档频率的特征选择方法的漏报率要高于基于信息增益的。基于信息增益方法的漏报率在 3%到 20%之间，而且平均漏报率低于 10%；而基于文档频率方法的漏报率在 6%到 27%之间，平均要高于 12%。

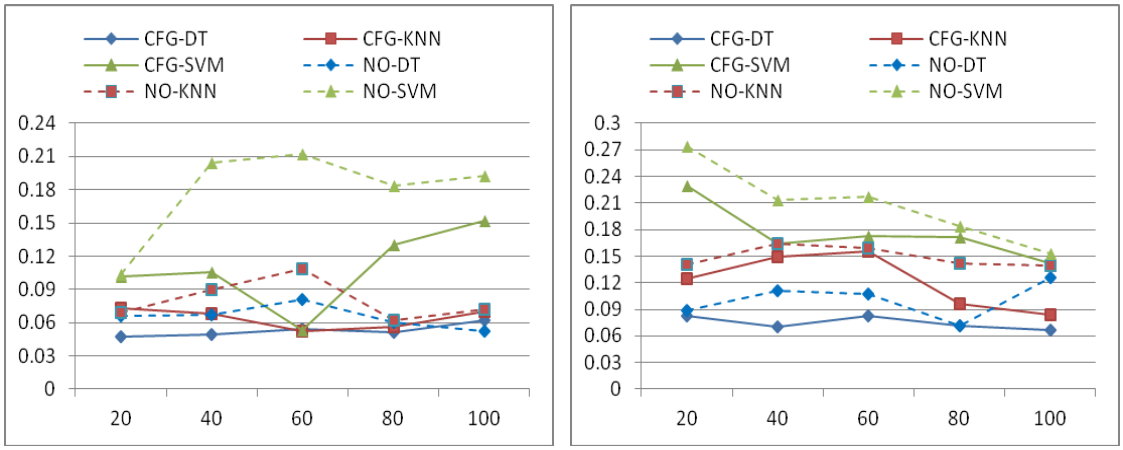


图 4-8 n=3 的漏报率对比图

通过对比图 4-9 中结果，当 n=4 时可以得出如下几点结论：

（1）在基于信息增益的特征选择方法中，总体上来看基于控制流程图的漏报率要低于基于文本的；但是特征数为 20，SVM 的漏报率正好相反。三种分类算法中决策树和 KNN 算法的漏报率较低，约为 3%到 10%，SVM 算法的漏报率较高约为 10%到 19%

（2）在基于文档频率的特征选择方法中，基于控制流程图的方法漏报率明显低于基于文本的，平均低 3%左右。三种分类算法中，决策树算法和 KNN 的漏报率较低平均约 6%到 13%，SVM 算法的漏报率较高约为 13%到 21%。

（3）总体来看，基于文档频率方法的漏报率要高于基于信息增益的。

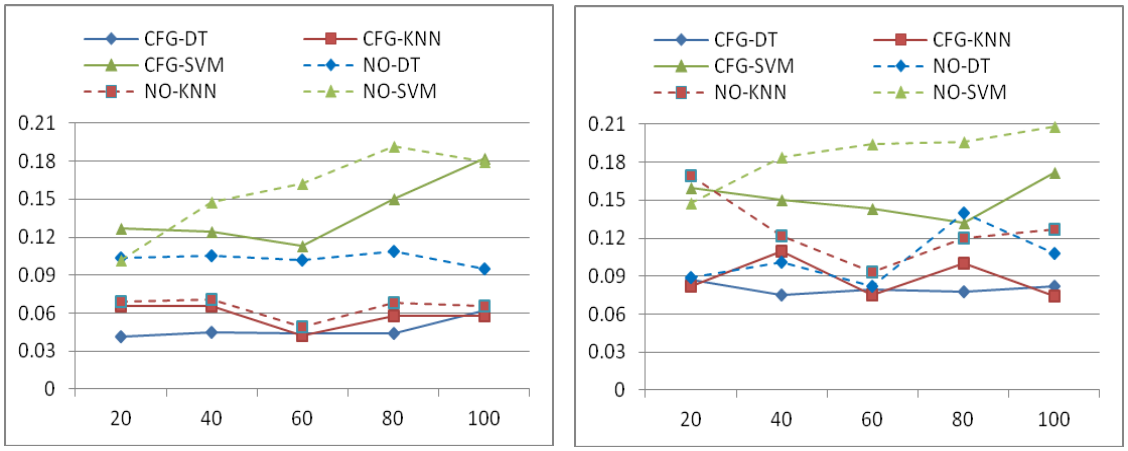


图 4-9 当 n=4 时的漏报率的对比图

通过对比图 4-10 中结果，当 n=5 时可以得出如下几点结论：

（1）在基于信息增益的特征选择方法中，基于控制流程图的漏报率要低于基于文本的，在特征数为 20 时 SVM 算法的结果相反，在特征数为 100 时决策树和 KNN 算法的结果相反；其中决策树和 KNN 算法的漏报率较低约为 3%到 9%，SVM 算法的漏报率较高约为 9%到 21%。

（2）在基于文档频率的特征选择方法中，基本上是基于控制流程图的漏报率较低。三种分类算法中决策树的漏报率较低，约为 6%到 12%，SVM 的漏报率较高，约为 12%到 21%。在特征数为 60 时，三种分类算法的漏报率相对较低。

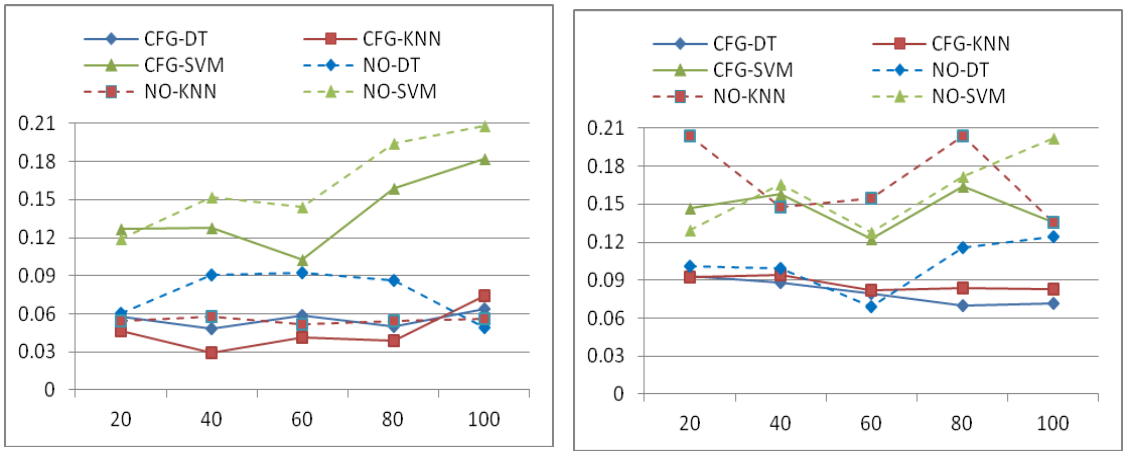


图 4-10 当 n=5 时的漏报率对比图

通过对上述三组图的比较和分析，可以发现基于文档频率的漏报率比较高，最高达到 27%。基于信息增益的相对较低约为 3%到 21%。总体上，基于控制流程图的漏报率要相对低一些。其中决策树和 KNN 算法的漏报率相对较低，而 SVM 算法的漏报率较高。整个实验的漏报率相对较高，因为误报率和漏报率有关，同一准确率，误报率低则漏报率就会相对高。

4.4.4 实验结果分析

通过上述实验结果的对比，可以得出如下结论：

（1）结合程序控制流程图的操作码序列提取方法的准确率、误报率及漏报率的效果要好于基于文本的。其中主要的原因是结合程序的控制流程图能很好的体现程序的行为特征，从而能更好的检测恶意代码。如图 4-11 所示为一后门程序的一段反汇编代码提取操作码序列的示意图，基于文本的操作码序列提取方法得到的序列是：xor, sub, nop,, jmp, add, add, xor, stosb，loop, jl, and, and, retf。基于控制流程图的操作码序列提取方法得到的序列如图中箭头所示，执行到 jmp 时跳转到 loc_40B689，接着继续跳转，这样提取的操作码序列为很多连续的 jmp。为了躲避检测，很多恶意程序有这样的特征。基于文本的提取方法只能按序提取操作码序列，而基于程序控制流程图的方法能很好的描述这种特征。

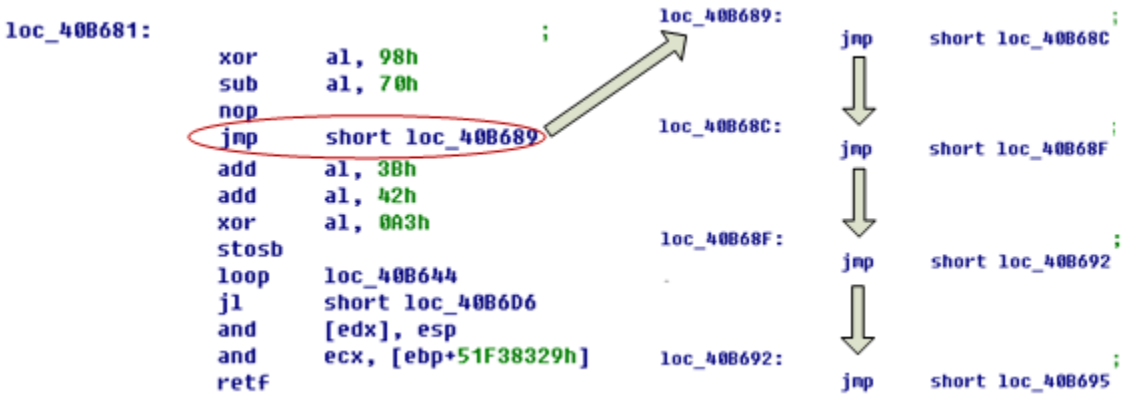


图 4-11 后门程序的一段反汇编代码提取操作码序列的示意图

（2）基于信息增益的特征选择方法的实验结果整体上比基于文档频率的好。三种分类算法中，决策树的分类算法结果较好而支持向量机的分类效果相对较差。特征数量的选取没有很明显的规律性。

4.5 本章小结

本章首先介绍了实验结果的评价方法和实验数据。其次，选取不同的特征长度 n 值 3、4 及 5 分别进行特征提取。然后，使用基于信息增益和文档频率两种方法选择不同的特征数量 20、40、60、80 及 100。接着，通过三种分类算法 K 近邻、决策树及支持向量机实现恶意代码的检测。最后，通过实验对比基于控制流程图的特征提取方法和基于文本的实验效果，根据准确率、误报率以及漏报率三个评价指标对比分析了实验结果，并做了相关的总结。

结　论

本文通过提取操作码序列作为特征分析了恶意程序，与传统的静态检测方法不同的是，本文不仅提取了操作码序列这一比较新颖的特征，而且在提取特征的时候考虑了程序的行为特征，按照程序的控制流程图提取每一条路径的操作码序列。将操作码序列作为恶意代码的行为特征，最终实现了恶意代码的检测。

本文的主要贡献如下：

（1)本文的创新点就是提出了将程序的控制流程图和操作码序列结合起来提取特征的方法。这样不仅考虑到程序的文本特征，也考虑到了程序的行为特征，为更好的检测恶意程序提供了保证。通过实验证明了基于控制流程图的提取特征方法，因为其考虑到了程序的行为特征，实验效果要比单纯的基于文本的方法好。对于两种特征选择方法，基于信息增益的特征选择方法的实验效果比较理想。在三种分类算法中，无论哪种特征提取和选择方法，决策树的实验效果较好，而支持向量机的效果相对差一些。从整个实验结果分析，本文提出的方法有较高的准确率，较低的误报率和漏报率。

（2）本文完成了恶意代码的查壳和脱壳工作，对脱壳后的恶意代码进行反汇编，分析反汇编后的文本，并在 vs2010 平台下结合 IDA SDK 开发了基于控制流程图和基于文本的操作码序列提取的两个插件。

（3）本文使用 n-gram 算法提取特征，设定不同长度的 n，提取相应的特征。采用了信息增益和文档频率特征选择的方法，选择一些分类能力好的特征。最后使用三种机器学习分类算法 K 近邻、决策树以及支持向量机实现了恶意代码检测系统。并根据不同的特征长度和不同的特征数量做了对比实验，并分析了实验结果。

虽然本文提出的方法取得了理想的实验效果，但是还有不足的地方有待改进和提高，具体分析如下：

（1）基于控制流程图的特征提取方法考虑到了行为特征，但是这样增加了特征的数量，会使得实验的计算量增加。可以考虑结合语义分析模型来降低实验的计算量。

（2）本文的实验结果中，漏报率稍微偏高。可以选择其他的特征选择方法，如 Fisher Score 及层次特征选择等方法，使得选择的特征有很好的分类能力。在特征提取时可以选择变长的 n-gram 算法，这样就不拘泥于固定的 n，可能会

提高实验效果。在分类时，可以考虑使用对特征加权的方法以便提高实验效果。

参考文献

[1] Boojoong K, Hye S K, Taeguen K, et al. Fast Malware Family Detection Method Using Control Flow Graphs[C]. ACM, 2011:1-6.

[2] McAfee, Best Behavior-Making Effective Use of Behavioral Analysis[C]. NETWORK ASSOCIATES, 2002:1-2.

[3] Christodorescu M, Jha S. Testing Malware Detectors[C]. ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '04)[C], Boston, MA, United States. 2004. Association for Computing Machinery, 2004:34-44.

[4] Garfinkel T, Adams K. Compatibility Is Not Transparency: VMM Detection Myths and Realities[C]. In Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems, San Diego, USA, 2007. USENIX Association:1-6.

[5] Ed S, Lenny Z.决战恶意代码[M]. 北京: 电子工业出版社 2005.展, 2005:1-10.

[6] Sung A H, Xu J, Chavez P, et al. Static Analyzer of Vicious Executables[C]. Proceedings of the 20th Annal Computer Security Applications Conference, Tucson, AZ, USA. IEEE Computer Society Press, 2004:326-334.

[7] Zhang B Y, Yin J P, Tang W S, et al. Unknown Malicious Codes Detection Based on Rough Set Theory and Support Vector Machine[C]. 2006 International Joint Conference on Neural Networks Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. July 2006:16-21.

[8] Trevor H, Peter S, Harald S. State Joining and Splitting for the Symbolic Execution of Binaries[J], Lecture Notes in Computer Science. 2009, 5779:76-92.

[9] Edmund M C, Doron A P, Model Checking[M]. MIT Press, 1999:5-23.

[10] Patrick C, Abstract Interpretation Based Formal Methods and Future Challenges[C]. In Informatics, 10 Years Back — 10 Years Ahead, R. Wilhelm (Ed.), Lecture Notes in Computer Science 2000. Springer, 2001:138-156.

[11] Christodorescu M, Jha S. Static Analysis of Executables to Detect Malicious Patterns[C]. In Proceedings of the 12th Conference on Usenix Security Symposium, Washington DC, USA, 2003. USENIX Association: 12-32.

[12] Christodorescu M, Jha S, Seshia S. Semantics-Aware Malware Detection[C]. 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2005.Institute of Electrical and Electronics Engineers Inc, 2005:32-46.

[13] Preda M D, Giacobazzi R. Control Code Obfuscation by Abstract

Interpretation[C]. The Third IEEE International Conference on Software Engineering and Formal Methods (SEFM '05), Koblenz, Germany, 2005. IEEE Computer Society: 301-310.

[14]Singh P, Lakhotia A. Static Verification of Worm and Virus Behavior in Binary Executables Using Model Checking[C]. Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society, West Point, New York, USA. IEEE: 298-315.

[15]Kinder J, Katzenbeisser S, Schallhart C, et al. Detecting Malicious Code by Model Checking[C]. In Proceedings of the Second International Conference on Detection of Intrusions and Malwares, and Vulnerability Assessment (DIMVA'05), Vienna, Austria, 2005. Springer Verlag:174-187.

[16]李佳静，梁知音，韦韬，毛剑. 一种基于语义的恶意行为分析方法[J]. 北京大学学报(自然科学版). 2008, 44(4):538-542.

[17]王晓洁，王海峰. 基于语义的恶意代码检测的算法研究[J]. 计算机系统应用. 2009, 8:103-106.

[18]孔德光. 结合语义的统计机器学习方法在代码安全中应用研究[D]. 2010:1-60.

[19]Tahan G, Glezer C, Elovici Y，et al. Auto-Sign: an Automatic Signature Generator for High-speed Malware Filtering Devices[J]. Comput Virol 2010 , 6:91-103.

[20]Tang Y, Xiao B, Lu X C. Using a Bioinformatics Approach to Generate Accurate Exploit-based Signatures for Polymorphic Worms[C]. Computers & Security 2009, 28:827-842.

[21]Wang L J, Li Z C, Chen Y, et al. Thwarting Zero-Day Polymorphic Worms With Network-Level Length-Based Signature Generation[C]. IEEE/ACM TRANSACTIONS ON NETWORKING.VOL.18.NO.1.FEBRUARY 2010: 53-65.

[22]Robert M, Clint F, Eugene B, et al. Unknown Malcode Detection Using OPCODE Representation[C]. In Proceedings of the 1st European Conference on Intelligence and Security Informatics (EuroISI). 2008:204-215.

[23]Chawla N, Japkowicz N, Kotcz A. Editorial: Special Issue on Learning from Imbalanced Data Sets[J]. SIGKDD Exploratuons Newsletter, 2004, 6(1):1-6.

[24]Schultz M, Eskin E, Zadok F, et al. Data Mining Methods for Detection of New Malicious Executables[C]. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001:38-49.

[25]Kolter J, Maloof M. Learning to Detect and Classify Malicious Executables in the Wild[J]. Journal of Machine Learning Research, 2006, 7:2721-2744.

[26]Mitchell T. Machine Learning[M]. McGraw-Hill, New York, 1997:12-23.

[27]Henchiri Q, Japkowicz N. A Feature Selection and Evaluation Scheme for Computer Virus Detection[C]. In: Proceedings of ICDM 2006, Hong Kong. 2006:891-895.

[28]Raja K S, Niklas L, Henric J. Accurate Adware Detection Using Opcode Sequence Extraction[C]. The Sixth International Conference on Availability, Reliability and Security. 2011:189-196.

[29]Dolev S, Tzachar N. Malware Signature Builder and Detection for Executable Code[C]. Patent Application. 2010:1-5.

[30]Sulaiman A, Ramamoorthy K, Mukkamala S, et al. Disassembled Code Analyzer for Malware (DCAM)[C]. In Proceedings of the IEEE International Conference on Information Reuse and Integration. 2005:398-403.

[31]Siddiqui M, Wang W, Lee J. Detection Internet Worms Using Data Mining Techniques[J]. Journal of Systemics, Cybernetics and Informatics. 2010, 6(6):48-53.

[32]Shahzad R K, Haider S I, Lavesson N. Detection of Spyware by Mining Executable Files[C]. In Proceedings of the International Conference on Availability, Reliability, and Security (ARES), 2010:295-302.

[33]Ismail B, Aitor G. Graphs, Entropy and Grid Computing: Automatic Comparision of Malware[C]. In Proceedings of the Virus Bulletin Conference, 2008:54-61.

[34]Halvar F. Structural Comparison of Executable Objects[C]. In Proceedings of the IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment, 2004:1-10.

[35]Igor S, Felix B, Xabier U P, et al. Opcode Sequences as Representation of Executables for Data-mining-based Unknown Malware Detection[J]. Inform.Sci, 2011:1-19.

[36]Perdisci R, Lanzi A, Lee W. Classification of Packed Executables for Accurate Computer Virus Detection[J]. Pattern Recognition Letters, 2008, 29:1941-1946.

[37]Perdisci R, Lanzi A, Lee W. McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables[C]. In Proceedings of the 23rd Annual Computer Security Applications Conference, 2008:301-310.

[38]Grimes R A. Malicious Mobile Code, Virus Protected for Windows[M]. O'Reilly& Associates, Inc, 2001:12-13.

[39]Skoudis E，Lenny Z， 陈贵敏. Malware：Fighting Malicious Code[M]. 北京: 电子工业出版社，2005:1-10.

[40]McGraw G，Morisett G. Attacking Malicious Code: A Report to the Infosex Research Council[C]. IEEE Software 2000(5):33-41.

[41]David S. Foundations of Computer Security[M]. Springer Press, 2006:1-10.

[42]Hoglund G, McGraw G. Exploiting Software: How to Break Code[J]. US: Addison Wesley, 2004:60-81.

[43]Shabtai A, Moskovitch R, Elovici Y, et al. Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-art Survey. Information Security Technical Report. Elsevier ltd. 2009:1-16.

[44]Eagle C,石华耀,段桂菊. IDA Pro 权威指南[M]. 北京:人民邮电出版社. 2010:1-10.

[45]Abou-Assaleh T, Cercone N, Keselj V, et al. N-gram-based Detection of New Malicious Code[C]. Computer Software and Applications Conference. Proceedings of the 28th Annual International. 2004(2):41-42.

[46]Han J W, Micheline K, 范明等. 数据挖掘概念与技术[M]. 北京:机械工业出版社. 2008:185-220.

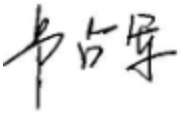
攻读硕士学位期间发表的论文及其它成果

1 Ding Yuxin, Dong Li, Zhao Bin, Lu Zhanjun. High Order Network with Self feedback to Solve Crossbar Switch Problem, ICONIP'2011, Shanghai, 2011.

哈尔滨工业大学学位论文原创性声明及使用授权说明

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于操作码序列的静态恶意代码检测方法的研究》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签名：  日期：2013 年 1 月 4 日

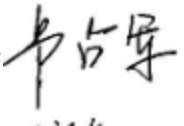
学位论文使用授权说明

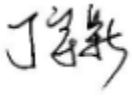
本人完全了解哈尔滨工业大学关于保存、使用学位论文的规定，即：

（1）已获学位的研究生必须按学校规定提交学位论文；（2）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（3）为教学和科研目的，学校可以将学位论文作为资料在图书馆及校园网上提供目录检索与阅览服务；（4）根据相关要求，向国家图书馆报送学位论文。

保密论文在解密后遵守此规定。

本人保证遵守上述规定。

作者签名：  日期：2013 年 1 月 4 日

导师签名：  日期：2013 年 1 月 4 日

致　谢

时间过得真快，转眼间就到了毕业的时刻，在哈工大深圳研究生院两年半的研究生生活是充实的、快乐的。研一，忙碌的上课生活不仅仅让我学到了很多知识，也让自己认识到很多不足。仍记得和同学忙碌而快乐的熬夜赶 project，还记得努力获得优异成绩的喜悦之情。研二，在老师和同学们的帮助下研究着自己的课题。在老师的悉心指导下我明确了研究方向，和同学们的讨论使我对科研越来越有兴趣。这两年半的生活我收获很多，在此我诚挚的感谢在生活中和学习中给我帮助和关心的老师和同学们，谢谢你们给我带来很多欢乐。

感谢我的导师丁宇新副教授。在科研上，丁老师严格要求我们，并耐心详细的指导着我们研究中遇到的各种问题。每周都会开会关心我们的科研进展并对后期的进展做详细的安排。非常感谢丁老师对我论文的指导。在生活中，丁老师和蔼可亲，关心我们生活的每个细节。这两年来，丁老师的责任感和严肃的科研态度深深影响着我，在此再次向丁老师表示衷心的感谢。

感谢智能计算中心的全体老师为我们提供了很好的学习条件。也感谢老师们组织的每周的羽毛球活动，既锻炼了身体又从中获得了很多欢乐，同时也增进了同学之间的友谊。感谢智能计算中心的所有同学，一起聚餐，一起打球，一起聊天，一起学习，相互关心，相互帮助等这些都给我留下了美好的回忆。在此也感谢我的舍友们，你们的帮助关心给我带来了很多快乐和学习的动力。找工作时大家的相互鼓励让我们都有了一个好的归宿，感谢你们激励着我一直前进。

感谢默默支持和关心我的父母和亲人，是你们用无私的爱一直守护着我直到现在。我会用一生去好好爱护你们，陪伴你们。感谢你们给我的一切。

感谢看雪学院和黑客防线等网站上热情的朋友。是你们的无私帮助使我的课题进展的很顺利，你们不仅仅教会了我很多知识，更让我懂的了时刻学习的重要性。我将在以后的生活中与你们共同关注安全方面的话题，共同学习，共同进步。

最后，感谢参加论文审阅的各位老师，感谢你们在繁忙的工作中，抽出宝贵的时间给我的论文提出宝贵的建议。谢谢你们。