1. **绪论**

**1.1 选题背景及意义**

**1.1.1 选题背景**

随着互联网的普及，恶意软件的危害也变得越来越来越难以控制。各种病毒、木马、蠕虫等恶意代码在网络间广泛传播，已经给个人、企业甚至政府带来了难以估量的损失。据国内知名互联网安全厂商奇虎360发布的《2016年中国互联网安全报告》显示，2016年全年，2016年360互联网安全中心共截获PC端新增恶意程序样本1.9亿个。敲诈者病毒在国内发生两次大规模传播，全国至少有497多万台用户电脑遭到了敲诈者病毒攻击。通过对受害者调研，42.6%的受害者不知道感染病毒的原因。预计在2017年敲诈者会增长10倍，且利用挂马攻击也将再次爆发。2016年360互联网安全中心共截获Android平台新增恶意程序样本1403.3万个，其中资费消耗类程序为74.2%。同PC端相似，手机端勒索软件也开始爆发，360全年截获新增手机勒索软件17万，170万台手机遭到攻击。在截获盗取个人信息的手机恶意程序样本中，67.4%的样本会窃取短信信息，34.8%的样本会窃取手机银行信息，10.0%的样本会窃取手机联系人信息，3.7%的样本会窃取手机通话记录，2.0%的样本会窃取社交软件（例如微信、QQ等）聊天记录，1.8%的样本会窃取手机录音信息，0.1%的样本会窃取手机照片信息。根据中国互联网安全报告显示，以下是2017年上半年5大典型安全威胁事件：

1. 2017年上半年“WannaCry”、“暗云Ⅲ”、“Petya”等多种类型的病毒木马连续集中的爆发，为社会和行业敲响了网络安全的警钟。《报告》显示，仅2017年上半年，腾讯安全反病毒实验室在电脑端总计已拦截病毒已超过10亿次，平均每月拦截木马病毒近1.7亿次，相较于2016下半年病毒拦截总量增长30%。其中，受“WannaCry”刺激，勒索类病毒仅第二季度就新增了13.39%，但“WannaCry”在非感染型敲诈类病毒占比中仅排第三，而带有感染传播方式的PolyRansom勒索病毒传播力更巨大，其占所有勒索类病毒的78.84%。
2. 国务院某App的H5遭遇流量劫持。5月中旬，某国字号的App遭遇流量劫持的传闻在业界流传。有消息称，该App某H5页面被植入色情内容广告，经排查“基本确定为用户当地运营商http劫持导致H5页面被插入广告……”。
3. 12306官方网站再现安全漏洞。4月21，有媒体记者发现在12306官方网站订票时发现，当退出个人账号，网站页面竟自动转登他人账号，且与账号相关联的身份证号、联系方式等个人信息均可见，随后记者在该页面点击常用联系人选项时页面再次刷新并显示他人账号及账号涵盖的所有信息。而记者尝试在网站账户页面的个人信息栏等其他选项进行操作，点击进入后均得到不同的个人身份信息。
4. 上亿优酷信息数据在暗网售卖。4月17日，外国媒体hackread报道，100759591条优酷账户信息数据库在暗网售卖，该数据库售卖价格定为比特币 0.2559，，人民币约 2065.56 元。
5. “土耳其犯罪家庭”的网络犯罪团伙掌握3亿苹果帐户。3月底，国外媒体报道，自称为“土耳其犯罪家庭”的网络犯罪团伙，通过电子邮件告知苹果公司他们掌握了超过3亿苹果帐户，并能远程清除所有装置的内容。他们宣表示只想苹果支付75000美元的比特币赎金，或者价值10万美元的iTunes礼品卡。若苹果公司在4月7日拒绝遵守他们的要求，他们将大量清除iCloud帐户。

在所有的网络安全事件中，尤以恶意代码的危害性最大，其带来的经济损失占很大比例。从八十年代初期开始出现第一个病毒开始，到如今恶意代码的不断发展壮大，入侵与防护的战役从未停止。纵观恶意代码发展历史，在利益的驱使下，无论其发展速度还是其破坏性都在不断增强，并且为了对抗杀毒软件的检测，各种抗查杀技术的应用，使得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发生了很大的变化，从以前的被动传播到如今的主动传播。由于现在出现了很多恶意代码的编制工具，并且极易获得，使得恶意代码制造成本越来越低，所以恶意代码的发布变得越来越频繁。恶意代码的发展，给个人、企业以及政府带来的损失时难以估量的，据统计80%以上的用户曾有意识或者无意识的遭受过恶意代码的侵袭。现如今，提高互联网安全已经成为国家的一项重要基本战略。

**1.1.2 选题意义**

很大一部分网络安全事件发生都是由恶意代码引起的，并且根据以往案例来看，往往都是恶意代码造成一定的损失之后，针对该恶意代码的分析及检测技术才会被提出。出现这种情况的原因无非有两个：首先是恶意代码越来越复杂，并且种类繁多，传播形式多种多样，使得用户很容易被感染；其次是恶意代码检测技术不够成熟，尽管很多学者为恶意代码检测做了很多研究，也提出了很多检测方法，但是理论到应用总是需要时间，而这些时间给了恶意代码去进一步变异的可能。此消彼长下，恶意代码的危害始终存在，而检测与反检测的斗争不会停止。

恶意代码能够在短时间内造成大范围传播的原因主要是信息共享技术和使用的普及。信息共享作为时代发展的必然产物，与人方便的同时，也给不法分子打开了新世界的大门，信息的快速流动加速了恶意代码的入侵。无论是Internet上的网页，还是光盘、U盘甚至接收到的Email都有可能携带恶意代码，防不胜防。可是时代的发展又离不开信息的共享技术，机遇与挑战并存，因噎废食是不明智也是不可取的。

近年来，恶意代码的使用与各种各样的经济甚至政治利益互相牵扯，其危害性和隐藏性日益增强，破坏的目标、目的以及要达成的后果更加具有针对性。而恶意代码的编写者也从最初的技术炫耀逐渐变为经济或者政治利益的追逐者。中国木马产业链一年的收入已逾上百亿元，黑色产业链正在逐渐成型。恶意代码的检测与反检测注定会是一场持久战。

综上所述，恶意代码的危害无处不在，不仅给个人、企业带来了巨大的损失，甚至可能给国家安全带来不可预期的危害。对于个人，恶意代码的入侵会导致个人隐私的泄露，造成经济或者名誉损失；对于企业来说，企业数据一旦遭到恶意代码的入侵，会导致企业大量数据资产外泄，给企业带来无可挽回的损失，甚至因此产生一些灰色的产业链；对于国家，信心安全是国家安全的重点组成部分，网路安全已经是国策和民生的大问题，信息安全成为国家战略，随着互联网的发展，信息安全的问题将会更加突出和重要。因此研究更加有效的恶意代码检测技术是非常具有现实意义的。

**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 本文研究内容**

本文设计并实现了一个自动化检测恶意代码的系统，并提出了一种新的静态恶意代码检测方法。主要是在程序的汇编代码中提取特征，然后根据特定的映射规则对汇编指令进行更高层次的抽象，以此更好的反映程序的行为特征。这种方法结合了基于特征码和行为的共同特征，最终使用机器学习的分类技术实现检测。这样既可以克服动态检测的不完整性，又解决了一般静态检测技术不能处理的混淆带来的困惑。首先，为了逃避病毒检测系统的检测，一般的恶意代码作者都会对恶意程序进行加壳处理，所以本文的第一步就是对恶意代码进行脱壳处理。其次，对恶意程序进行反汇编处理，得到程序的汇编代码，并提取汇编操作码序列。然后，配置特定映射规则对汇编操作吗序列进行抽象化处理。接着，使用n-gram算法提取特征并且根据配置文件选择降维方法，进行特征选择。最后，使用随机森林、支持向量机以及K邻近三种机器学习分类算法实施分类。本文的工作有以下几点：

第一：本文提出一种新的方式对汇编代码进行抽象，以此在高层次反映程序的控制流。这样可以在一定程度上避免混淆技术带给静态检测技术的困惑。

第二：采用模块化编程的技术实现检测系统，在映射规则、n-gram算法n的选取以及降维算法的选取上，采用配置文件的方式，方便了实验的进行和结果的对比。

第三：对不同的映射规则下得到的结果进行对比，然后总结出最优的映射规则。

**1.4 本文章节安排**

本文总共分为四章：

第一章为绪论，主要介绍选题背景以及选题的意义，并且详细阐述了国内外对于恶意代码检测技术的研究现状和存在的问题，最后介绍了本文的研究内容和章节安排。

第二章是恶意代码检测技术概述，主要介绍恶意代码的定义、分类和现有恶意代码检测技术的原理和优缺点。

第三章是基于机器学习算法的恶意代码检测技术，按照步骤详细介绍了恶意代码的查壳与脱壳、反汇编技术、汇编码抽象方法以及中间码序列的提取与选择。然后分析中间码序列特征以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。

第四章是系统架构的设计以及实验结果。介绍整个系统的架构设计以及整个系统的核心模块和各项实验结果，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。

**第二章 相关理论与技术**

**2.1 恶意代码简介**

**2.1.1 恶意代码的定义**

恶意代码也成为恶意软件，是对各种敌对和入侵软件的概括性术语。包括各种形式的计算机病毒、蠕虫、特洛伊木马、勒索软件、间谍软件、广告软件以及其他的恶意软件。形式上多种多样，可以是可执行文件、脚本、插件等等。其违背使用者的意愿去执行一些操作，损害用户的利益以达到入侵者不可告人的目的。

**2.1.2 恶意代码的分类**

根据不同的依据，恶意代码有很多种不同的分类方法，没有一种标准的分法，但是常见的种类有：计算机病毒、蠕虫、特洛伊木马、间谍软件、勒索软件等等。下面对几种恶意代码做简要介绍：

(1)计算机病毒。病毒是早期产生的最主要的恶意代码之一，病毒是能够自我繁殖并寄生在其他程序中的代码，这个被寄存的程序被称为宿主程序，但是病毒不能单独运行，必须通过激活宿主程序并满足一定条件下，病毒就能干扰电脑正常工作，扰乱或破坏己有存储的信息，甚至引起整个系统不能正常工作。一般而言计算机病毒通常由三个单元和一个标志构成：引导模块、感染模块、破坏表现 模块和感染指标。1、引导模块是指将计算机病毒感染的宿主程序设法引导安装到

计算机操作系统中，为以后的感染、破坏两个后期模块提供前期的有效准备，一般而言不同的计算机病毒有不同的引导操作，而且引导操作往往是隐蔽的，不易被用户察觉和发现的。2、感染 模块包括两个部分，一个是用来激活感染功能的判断部分。该模块提供一个感染

的标志，用来判断计算机是否被感染。另一个是执行感染功能部分。这一部分主要的功能就是监控宿主满足条件的时机，并及时的将计算机病毒存入到系统特定的位置。3、破坏表现模块与感染模块一样包括两个部分，一是具有触发破坏表现功能的判断部分。二是具有破坏表现功能的实施部分。计算机病毒一般具有寄生 性、传染性、隐藏性、破坏性、潜伏性等特征。

（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。

**2.2 恶意代码的检测技术**

目前用于商业的恶意代码检测软件中，一般采用的都是基于“特征码”的检测技术，基本思想是，当新的恶意代码被发现后，对其进行采集取样，分析代码构成，提取有用的特征码，然后将新的特征码加入已有数据库中，用户更新病毒库之后，就会使用新的特征库去匹配恶意代码，如果匹配成功则进行相应的处理。但是特征码检测技术的缺点是只能对已知恶意代码进行有效的将测，对与未知或者稍加变动的恶意代码无能为力。因此，在恶意代码检测领域提出了启发式检测算法来预防和检测新的恶意代码。根据对恶意代码分析原理的不同对现有恶意代码检测方法进行分类，主要分为基于特征码的检测技术、基于行为的检测技术、基于启发式的检测技术、基于语义的检测方法和基于机器学习算法的检测技术等。下面将详细介绍几种检测技术。

1. 基于特征码的检测技术

基于特征码的检测方法是使用最古老和最广泛的方法。被Symantec等多有著名病毒检测厂商所使用，是目前已知的所有恶意代码检测方法中最简单、开销最小的方法，广泛用于文件类型的病毒检测中。检测软件的核心就是恶意代码特征库的完全性，当需要扫描某个程序是否有恶意企图时，启动特征扫描提取特征，然后再与特征库进行匹配，如果匹成功，则判断该程序是恶意的。此技术的关键在于如何选取最能代表恶意程序的特征值。采用该方法，检验结果准确，鲜有误报情况，但该方法对于未知或者变形恶意代码无能为力。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。

1. 基于行为的检测技术

基于行为的检测方法是利用恶意代码的特有行为来检测恶意代码的方法。恶意代码的行为有相对的稳定性和已于检测的特点，比如特定的系统调用，恶意代码要完成自身逻辑功能，即完成对系统的入侵和破坏，就必须获取系统非法权限，调用系统的资源【11】，这样通过分析恶意代码的行为就可以方便的分析检测出恶意代码。当程序运行时，监控其行为，如果发现了异常行为，则立即报警。一般用于检测恶意代码的行为特征如下：

* 对特定文件执行写操作：有些恶意代码时依附而生，所以在其执行时，就要将自身代码附加在感染文件中，可以监控是否有异常写操作。
* 监控系统调用序列：某些系统调用序列可以体现某种程度的程序语义。系统调用是用户态和内核态的唯一接口，恶意代码想要获取高级权限实施破坏行为，就必然要经过系统调用接口。
* 修改内存总量：恶意代码为了完成特定的恶意意图，经常会常驻在内存中，并且不能被覆盖，那么将会减少系统内存的总量，使得该段内存不受系统内核控制。

1. 基于启发式的检测技术

启发式检测方法是对恶意代码特征提前设定一个阈值，在对文件进行扫描后，当提取的特征和恶意代码特征的相似度达到一定的值，这认定该文件是恶意代码。例如一些恶意代码都会固定的对一些内核函数进行调用，通常这些调用的顺序是有一定的规律，因此利用对内核函数的名称和调用次数进行分析，可以构建一个恶意代码对内核函数的特征。启发式方法属于主动防御技术，对未知的恶意代码检测具有明显的效果，因此，这种方法在现如今的商业开发被重点应用。启发式检测可分为静态启发和动态启发。

静态启发方法其实是对传统的特征识别方法的一种扩展，通过分析程序对系统API的调用序列作为特征，有领域专家根据自身经验，研究总结出某些恶意代码的行为特征，当对行为进行监控时，此类特征一旦被发现，就立即报警并做相应的处理。这种方法能够有效的检测出已知的恶意代码，并发现部分未知的恶意代码，但在发现恶意代码的时候，系统往往已经被感染。另外，行为检测是对系统进行实时的监控，因此可能会持续占用大量内存、CPU等系统资源。在商业领域中，该方法主要用于辅助性检测。

动态启发式技术主要的工作原理是在计算机系统中划分出一各独立的虚拟环境，当发现可疑程序时，并不立即停止，而是让其继续运行。“沙盒”技术就是动态启发式技术的一种，沙盒会对可疑程序的行为进行记录，直到恶意代码完全暴露后，它在执行回滚操作，使计算机恢复到执行可疑程序之前的状态。近年来病毒检测厂商已经将沙盒技术应用与商业的查杀工具中并进行了推广。

1. 基于语义的检测技术

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。

1. 基于机器学习的检测技术

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。

**2.3 恶意代码分析技术**

恶意代码分析是确定恶意代码意图的过程，是实行恶意代码检测的必要前提。恶意代码分析的直接结果是用于实现恶意行为建模的元数据信息，如指令流、API调用序列等，为后续恶意代码的检测工作提供必要的支持。

恶意代码的分析技术一般可分为静态分析和动态分析。

1. 静态分析技术

静态分析技术是指对被测软件的源程序或者二进制码进行扫描，从语法、语义的层面去理解程序的行为，以期望获取程序在运行过程中的信息，而不需要运行程序。

要进行恶意代码的静态分析，首先需要对恶意程序进行反汇编，常用的反汇编工具有：W32DASM、objdump、PEid、HIEW、IDA Pro等。

静态分析技术由于不会运行程序，因此不会对计算机系统造成任何伤害，其分析效率相对动态分析而言较高，同时由于静态分析技术从程序本身入手，因此可以获得程序的全部信息，分析结果较为全面。但是由于静态分析技术的前提条件是对程序进行正确的反汇编，现如今很多恶意代码编写者常常会对恶意代码进行加壳、加密或者压缩使得恶意程序很难被正确的反汇编。总之，如果恶意代码无法被正确的反汇编，那么静态分析将会失效。

1. 动态分析技术

动态分析技术是指在可控环境下实际运行程序，监控执行过程中的程序行为，记录程序执行的信息。由于动态分析需要先运行程序，所以为了防止恶意代码对当前环境的破坏，系统在普或者恶意代码相关信息之后，会自动恢复到恶意代码执行前的最初状态，防止影响下一次的分析结果，但是动态分析技术能够获得恶意代码执行时的真实信息，可以有效地解决静态分析中譬如加壳、加密的干扰。

当前最流行的动态分析技术是动态污点分析技术【39】，它的基本原理是将一切不信任的外部数据标记为污点，然后跟踪标记为污点的数据的传播情况，并记录相关的系统调用或者指令执行等相关信息，然后以此信息进行检测。动态污点分析能够记录恶意代码更细粒度的精确特征，是当前非常热门的恶意代码检测技术。

动态分析技术也存在缺陷，比如开销大，一次只能分析一条路径，恶法应对恶意代码存在多路径的问题，同时由于恶法模拟出一个完全真实的计算机环境，对某些环境敏感的恶意代码无法进行有效的检测，因为恶意代码能够检测到虚拟机或者仿真机存在的情况，从而隐藏自身的真实行为，也无法知道某些恶意代码何时才会触发，动态分析技术也会受到行为层的混淆技术的干扰【40】如等价行为替换、模拟序列或者混淆序列等。于是有学者开始尝试静态分析与动态分析相结合【41】的方式进行恶意代码的检测，充分利用两种分析技术的优点。

两种分析技术各有优缺点，静态分析技术开销小，关注的是恶意程序本身的语法或者结构特征，动态分析技术开销大，关注的是恶意代码的行为特征，各有侧重点。静态分析技术分析全面，可获得恶意代码的全部信息，但获取特征的方式一般都是无导向的，因此可能包含大量无用信息，也易受代码迷惑技术的影响，动态分析技术能够获得程序的真实行为信息，但一次只能获得一种行为并且与当前的检测环境相关，信息不够全面。当前应用最为广泛的技术还是静态分析技术。总之，无论是静态分析还是动态分析，都需要借助恶意代码分析技术和监控技术获得恶意代码的基本属性和执行信息，以便深入理解恶意代码的功能，进一步实现恶意代码的检测和抑制。

**2.4 恶意代码反检测技术**

恶意代码检测与反检测技术总是相互促进，相辅相成。检测技术的进步也带动了反检测技术的发展，当前出现了各种各样的恶意代码反检测技术，现总结如下：

1. 加壳技术

恶意代码作者为了防止自己的程序被检测软件发现，利用一些软件技术给恶意代码加外壳，可以是利用算法将自己伪装成正常程序，或者利用特殊的算法将自己压缩或加密，使得检测软件很难检测。但是这些“壳”都有一个特点就是，他们先于程序获得执行控制权，然后把伪装后的程序还原，再把执行权交还给原始代码：是一类自修改代码。

1. 反虚拟执行技术

不可否认，虚拟执行的系统和真实系统或多或少存在差异[36]。比如，硬件上，调试器总是会设置硬件断点，而虚拟机总是在模拟硬件，这和真实的硬件是有差别的；执行环境，内核地址空间，对于虚拟机和真正的机器是不同的，还有调试器必须挂靠某些进程来插桩进程用于监控；应用程序，虚拟机和调试器都有外部应用程序，对进程可见，用于检查运行环境。一些指令在虚拟机环境中，执行时间总是远远长于真实环境，一个经常执行此类指令的程序能够指示它在虚拟机环境中运行。

1. 代码迷惑技术

恶意代码迷惑技术是指通过某种程序代码变换，改变自身在空间和时间上的结构，但是完成相同的逻辑功能【32，33】。恶意代码在进行迷惑处理之后，使得逆向工程分析变得难以进行。迷惑技术本身是一种保护软件的手段，但是常常被用来对抗分析和检测。恶意代码的迷惑技术可以有效的对抗恶意代码的静态分析技术和动态反汇编技术。目前主要有基于加密的迷惑技术和基于代码变换的迷惑术。其中代码变换主要指在源程序中，利用等价指令替换、指令位置交换、添加新指令等手段改变程序形式，但逻辑功能保持不变。

**2.5 本章小结**

本章首先介绍了恶意代码的定义以及恶意代码的分类，具体介绍了病毒、特洛伊木马、蠕虫等恶意代码的特征和危害。其次，介绍了恶意代码的检测技术，详细阐述了基于特征码的检测技术、基于行为的检测技术、基于启发式的检测技术、基于语义的检测技术和基于机器学习的检测技术。最后，针对目前主流的恶意代码分析技术以及反检测手段做了详细说明。

**第三章 基于机器学习算法的恶意代码检测技术**

**3.1 引言**

本文研究了基于机器学习算法的恶意代码检测技术，通过对汇编操作码不同程度的抽象，以此来勾勒出程序的行为轮廓，并实现对恶意代码的检测。研究方法的主要步骤是：首先，数据预处理，对所有的恶意程序进行脱壳处理并对脱壳后的程序进行反汇编，得到其汇编文本文件；其次，根据上一步骤得到的汇编文本文件提取汇编操作码，并根据映射关系对汇编操作码进行抽象，得到三种不同抽象级别的中间码，记为Level-0、Level-1和Level-2，抽象程度由低到高；然后，分析各个类别恶意代码的中间码序列特征，分别根据2-gram、3-gram、4-gram算法提取中间码序列特征，接着使用数据挖掘技术对各个抽象级别的中间码进行关联分析，可以证明各个类别的频繁项集具有明显区别，所以本文有理由使用概率矩阵作为机器学习的输入；最后，使用随机森林、支持向量机和K近邻算法实现对恶意代码的分类检测。下面将以此对每一个步骤进行详细介绍。

**3.2 数据预处理**

**3.2.1 查壳与脱壳**

壳是指在计算机软件中一段专门保护软件不被非法修改或反编译的程序。它们一般都先于程序运行，拿到控制权，然后完成保护软件的功能。

壳通常分为两类：压缩壳和加密壳。压缩壳出现较早，可追溯到DOS时代，使用压缩壳可以帮助缩减PE文件大小，隐藏PE文件内部代码和资源，便于网络传输和隐藏。压缩壳通常有两种用途，一种是单纯用于压缩普通的PE文件，另一种则会对源文件产生较大的变形，严重破坏PE文件头，通常用于压缩恶意程序。常见的压缩壳有：Upx、ASpack、PECompat。加密壳，也称为保护壳，它主要的功能是防止逆向分析技术，保护PE文件不被逆向分析。加密壳保护的文件通常比PE源文件大得多。常见的加密壳有：ASProtector、Armadillo、 EXECryptor、Themida、VMProtect。

现阶段主要的查壳手段有两种：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。

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。

**3.2.2 反汇编技术**

反汇编指将机器代码转换为汇编代码、低级转高级的意思，常用于逆向工程领域。目前网络上许多“免费软件”，PSP、PS、NDS游戏机的破解和苹果系统的越狱都跟反汇编息息相关。

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。

反汇编工具有很多，如有 IDA Pro、C32Asm、W32DASM、花指令清除器 1.2 等等。IDA Pro 是一款专业的反汇编工具，是由 DataRescue 开发的现己成为了很多 Hacker，Cracker， Reverse engineerer 的必备工具。C32Asm 的特点是集反汇编、 16 进制工具 、Hiew 修改功能于一体。静态分析中文软件的利器的优点是速度 快。花指令工具可以去除花指令，虽然目前效果不是太好，但目前也是对付静态 分析的一个重要手段。

本文使用objdump工具对恶意代码进行反汇编，主要使用objdump -d指令，环境是Windows下进行。

**3.2.3 操作码提取**

本实验使用的数据包含两部分，第一部分是所有的恶意程序反汇编后的文件，文件名为对恶意程序使用MD5哈希算法之后得到的字符串；第二部分是一个label.csv文件，包含两列，第一列为恶意程序的MD5字符串，第二列为该恶意程序所属类别。操作码提取流程如图x:



图x

首先读入label.csv文件，存储文件名和对应类别的映射关系；然后读取所有的恶意代码文件名，接着根据配置文件中的线程数，启用相应多的线程，每个线程处理一部分汇编文件，提取出对应的操作码；接着根据先前读入的文件名和类别映射关系，或者每一个文件所属类别；最后将提取出的操作码文件放置在对应类别的文件夹下。

对于操作码提取的具体过程，本文使用了两个小技巧。第一，使用了配置文件中的Level-0级别的抽象映射关系，由于Level-0级别实际并未对汇编操作码做任何程度的抽象，即就是所有汇编操作码，所以可以用于操作码的提取，并且该映射关系存储类型为unordered\_map，所以查找复杂度近似为,不会带来任何性能上的问题。第二，使用了多线程处理操作码提取，具体线程数可根据实际环境进行动态配置，进一步提高了处理速度。

**3.3 操作码抽象及特征分析**

**3.3.1 操作码抽象化**

将汇编操作码先进行抽象化处理是很多研究常用的手段之一。本文将操作码进行抽象化主要有两点考虑：

1. 汇编语言是面向机器的程序语言，是一种用文字助记符来表示机器指令的符号语言。它是目前所有编程语言中最接近机器码的一种语言。它的针对性特别强，需要对机器硬件进行精确的控制，所以它的每一条指令都是极致细化的，这也导致了汇编操作码的局限性。当使用n-gram算法时，一般来说n的取值不会太大，n条少量的汇编操作码并不能反映出程序行为的一般特征。如果对汇编操作码进行抽象，得到更高维度抽象的中间码，同样是n条中间码，其反映的行为特征将会更加明确及有意义，对于恶意代码的检测是存在积极意义的。
2. 将汇编操作码进行抽象化处理能够显著的提升实验的效率。本文使用概率矩阵作为恶意代码的特征，即机器学习的输入，而x86汇编操作码的种类就有100多种，当使用2-gram、3-gram、4-gram算法时，其特征维度分别是1002、1003、1004，即使后续对概率矩阵进行降维操作也是非常低效的。根据查阅文献以及对大量的汇编操作码进行分析后，本文分别对汇编操作码进行两种级别的抽象，抽象只有中间码的种类分别为9种和5种，其特征维度数的底数将会是9和5，这将显著提升实验效率，并且由实验分析及结果也知，这种抽象操作对恶意代码的检测是有积极意义的。

本文对汇编操作码进行了三种级别的抽象，分别是Level-0，Level-1，Level-2。对于Level-0，表示未作任何程度的抽象，即汇编操作码原本的形式；Level-1和Level-2分别对汇编操作码进行了不同程度的抽象精简。下面分别对Level-1和Level-2进行详细介绍。

Level-1将汇编操作码映射为9类中间码：

1. 数据传输，比如：MOV、PUSH、POP、IN、OUT等；
2. 算术运算，比如：ADD、SUB、CMP、DIV、MUL等；
3. 逻辑运算，比如：NOT、AND、OR、TEST等；
4. 移位运算，比如：SAL、SHL、SAR、SHR等；
5. 字符串操作，比如：MOVS、STOS、CMPS等；
6. 过程控制，比如：JMP、JZ、JP等；
7. 标志位控制，比如：CLC、STD、STC等；
8. CPU控制，比如：HLT、WAIT、NOP等；
9. 其他，伪指令及不常用的指令，比如：SEGMENT，ASSUME，END等。

下表表示出这9种中间码的表示方法：

Level-1抽象中间码的表示

|  |  |
| --- | --- |
| 种类 | 表示方法 |
| 数据传输 | DATA\_TRANS |
| 算术运算 | ALTH\_OPE |
| 逻辑运算 | LOGIC |
| 移位运算 | SHIFT |
| 字符串操作 | STR\_OPE |
| 过程控制 | PRO\_CTRL |
| 标志位控制 | FLAG\_CTRL |
| CPU控制 | CPU\_CTRL |
| 其他 | OTHER |

Level-2将汇编操作码映射为5类中间码：

1）数据传输，比如：MOV，PUSH，IN等等；

2）运算，其中包含算数运算指令(ADD、 SBB等)和逻辑运算指令(OR，XOR，TEST等等)；

3）程序转移，这是指影响程序正常流程的 指令，比如：JMP，CALL，RET，INT等；

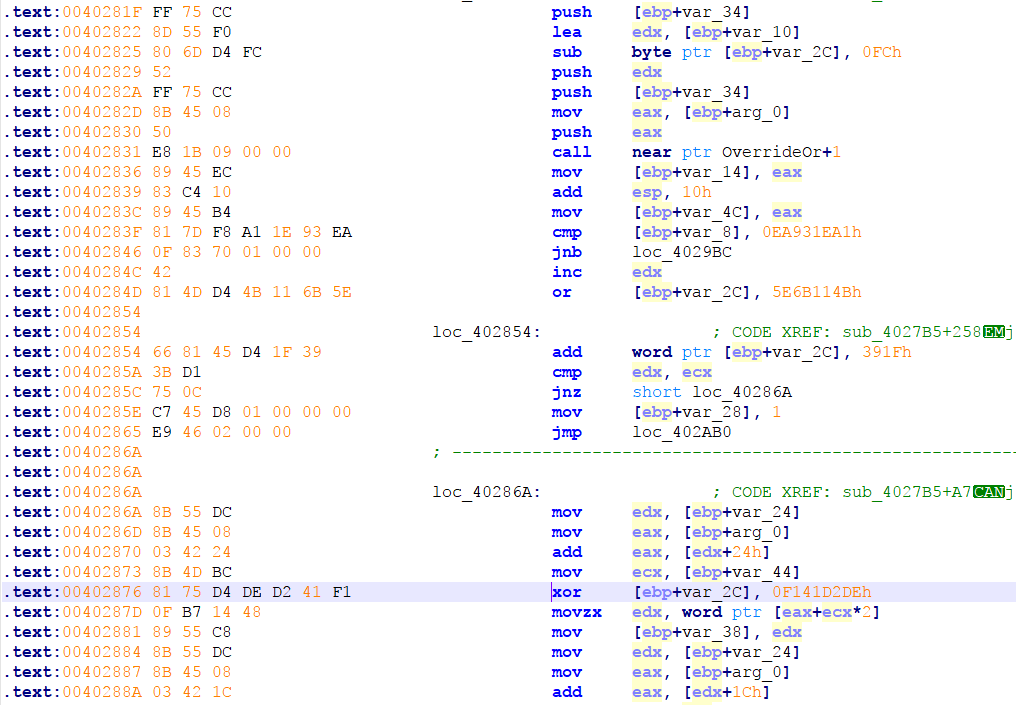
4）处理机控制，比如：NOP，HLT，WAIT 等等；

5）其他，伪指令及不常用的指令，比如：SEGMENT，ASSUME，END等。

下表表示出这5种中间码的表示方法：

|  |  |
| --- | --- |
| 种类 | 表示方法 |
| 数据传输 | ASSIGN |
| 运算 | TEST |
| 程序转移 | JMP |
| 处理机控制 | CTRL |
| 其他 | OTHER |

下图x是截取样本种的一段汇编程序，为节省篇幅，本文分别使用Level-0、Level-1、Level-2对其前15条汇编操作码进行抽象，翻译后的中间码顺序分别表示为下：



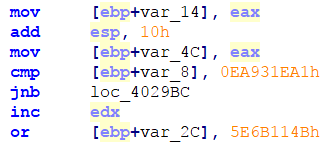
Level-0：push->lea->sub->push->push->mov->push->call->mov -> add->mov->cmp->jnb->inc->or

Level-1：DATA\_TRANS-> DATA\_TRANS-> ALTH\_OPE-> DATA\_TRAN S-> DATA\_TRANS-> DATA\_TRANS-> DATA\_TRANS-> PRO\_CTRL-> DATA\_TRANS-> ALTH\_OPE-> DATA\_TRAN-> ALTH\_OPE-> PRO\_CT RL-> ALTH\_OPE-> LOGIC

Level-2：ASSIGN-> ASSIGN-> TEST-> ASSIGN-> ASSIGN-> ASSIGN -> ASSIGN-> JMP-> ASSIGN-> TEST-> ASSIGN-> TEST-> JMP-> TEST-> TEST

**3.3.2 n-gram算法提取特征**

n-gram是计算机语言学和概率论范畴内的概念，目前n-gram被广泛应用于自然语言的自动分类功能。该模型基于第n个词出现只和其前n-1个词相关的假设，对于处理依靠序列关系分类的问题具有很强的优势。本文应用n-gram参考文献）算法的思想，将获得的中间码序列，设置不同的n的大小，获得一系列滑动窗口为n的中间码序列。以下图为例。



定义一个程序为p,操作码序列为o，则可表示程序，其中m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。对于图x中给出的操作码序列，应用3-gram算法可获得特征为：，，，，，然后根据不同的抽象级别得到其对应的n-gram表示。对于n-gram算法，很难确定滑动窗口n的大小，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2、3以及4来提取相应的特征。

**3.3.3 特征分析**

应用n-gram算法分别提取2-gram、3-gram和4-gram特征，并根据不同抽象级别做映射操作，得到最终提取到的特征文本。本节将使用Eclat算法分析各类别特征文本的频繁项集，并最终证明概率矩阵可作为区分不同类别恶意代码的特征。

本文使用Eclat算法来分析特征文本的频繁项集。尽管Apriori算法是最广为人知的关联规则挖掘算法，但是算法需要多遍扫描数据库因而会产生大量的候选项集，支持度的计算也很耗时。相对于Apriori算法，Eclat算法采用了等价类、深度优先遍历、求交集等策略，支持度计算效率有很大改善。Eclat算法采用了垂直数据结构来存储每个事物的ID（tid）列表，在计算k+1-项集的支持度时，Eclat算法只需要计算两个k-项集的tid-表的交集即可。最后，Eclat算法也从频繁项集中生成关联规则。

本文使用频繁项集的平均相似度评估频繁项集对于恶意代码分类的效果，平均相似度越低，代表频繁项集分类效果越明显，反之，代表分类效果差。本文假设频繁项集平均相似度小于50%便可用于恶意代码分类。频繁项集的平均相似度定义如下：

式中分别都是集合，A表示集合的集合， 表示集合A的基数。例如，集合，其中集合频繁项集为，集合频繁项集为 ，集合频繁项集为，那么

本文选取最频繁的10个项集来计算平均相似度，结果如图x：

图x 平均相似度对比图

由图可以看出，任意抽象级别，随着n-gram中n的增大，频繁项集平均相似度越来越低；对于2-gram，随着抽象级别的增大，频繁项集平均相似度增大，3-gram中，抽象级别增大，频繁项集平均相似度减小，4-gram中，抽象级别Level-1的频繁集相似度最大，但是也未超过30%；频繁项集平均相似度最低为23.33%，抽象级别Level-2并且n-gram中n值为4；除了抽象级别Level-2中2-gram的相似度超过50%之外，其他各个类之间的频繁项集平均相似度均未超过50%。由此，本文有理由相信根据频繁项集来对恶意代码进行分类是可行的。

**3.3.4 概率矩阵**

由3.3.3小节可知，不同类别的恶意代码的频繁项集是可区分的，并且不同抽象级别的区分度也是不同的，本文针对每个特征的出现频率构建一个概率矩阵，以此作为机器学习算法的训练集和测试集。

以抽象级别Level-1为例，对图x应用2-gram算法，可得到，统计每个词组出现的次数，然后除以总的词组数6得到各个词组的频率，将该频率作为该段程序的概率分布中的值，表x给出了图x程序片段的概率分布：

表x 2-gram&Level-1概率分布（未填写表示0）

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | DATA\_  TRANS | ALTH\_  OPE | LOGIC | SHIFT | STR\_O  PE | PRO\_C  TRL | FLAG\_  CTRL | CPU\_C  TRL | OTHE  R |
| DATA\_TRANS |  | 2/6 |  |  |  |  |  |  |  |
| ALTH\_OPE | 1/6 |  | 1/6 |  |  | 1/6 |  |  |  |
| LOGIC |  |  |  |  |  |  |  |  |  |
| SHIFT |  |  |  |  |  |  |  |  |  |
| STR\_OPE |  |  |  |  |  |  |  |  |  |
| PRO\_CTRL |  | 1/6 |  |  |  |  |  |  |  |
| FLAG\_CTRL |  |  |  |  |  |  |  |  |  |
| CPU\_CTRL |  |  |  |  |  |  |  |  |  |
| OTHER |  |  |  |  |  |  |  |  |  |

然后将这81种组合作为概率矩阵的列，每一个程序的概率分布作为概率矩阵的一行，由此构建本文所诉的概率矩阵。

设集合，表示抽象级别。对于本文，，其中对于，x代表所有的汇编操作码，大约140种；对于，x表示9种中间码中任意一个；对于，x表示5种中间码的任意一个。设，即集合的基数，所以。易知，对于2-gram算法，概率矩阵的列数等于的笛卡儿积的基数，即就是；对于3-gram算法，概率矩阵的列数等于三重笛卡儿积的基数，即就是；对于4-gram算法，概率矩阵的列数等于三重笛卡儿积的基数，即就是。归纳总结可知，对于n-gram算法，概率矩阵的列数就是的n重笛卡儿积的基数。

本文提取的汇编操作码共有140多种，不妨设为100种，本文所用恶意代码数为10868，则概率矩阵行数不妨设为10000，假设矩阵中元素使用4字节的float类型存储，那么对于Level-0抽象级别下，2-gram、3-gram、4-gram分别需要的内存数如表x:

表x Level-0抽象级别n-gram所需内存

|  |  |
| --- | --- |
| n-gram | 概率矩阵所需内存 |
| 2-gram |  |
| 3-gram |  |
| 4-gram |  |

所以本实验提出的方法有一定的适用范围：设内存大小为字节，中间码种类数为C个，n-gram算法n取值为N，概率矩阵存储类型大小为T字节，那么只有当满足时才可使用本文提出的方法进行恶意代码检测。本文接下来只关注Level-1和Level-2抽象级别的分类检测效果。

**3.4 恶意代码分类**

**3.4.1 随机森林**

随机森林是一种集成学习算法，该算法在学习过程中将产生多个决策树，每棵决策树会根据输入数据集产生相应的预测输出，算法采用投票机制选择类别众数作为预测结果。

随机森林算法的目标是通过将多个弱学习机（如单棵决策树）组合得到一个强学习机。算法处理过程与bagging算法非常相似，不同的是随机森林算法包括两个参数：ntree（决策树个数）和mtry（可用来寻找最佳特征的特征个数），而bagging算法只有ntree参数，当随机森林的mtry设置成与训练数据集的特征值一样大时，随机森林算法就等同于bagging算法了。

假设当前拥有N个特征数为M的样例，首先采用bootstrap对数据进行采样，每次随机采样N个样本作为单个决策树的训练数据集。在每个节点，算法首先随机选取m（m<<M）个变量，从它们中间找到能够提供最佳分割效果的预测属性；然后，算法在不剪枝的前提下生成单个决策树；最后从每棵决策树都得到一个分类预测结果。如果是回归分析，算法将所有预测的平均值或者加权平均值作为最后输出，若果是分类问题，则选择类别预测众数作为最终预测。

训练模型的生成流程如图x所示，检测模型流程如图x所示。

图4 随机森林训练模型流程图 图5 随机森林检测模型流程图

本文使用R语言中的caret算法包完成随机森林算法的调参及交叉验证，使用概率矩阵作为模型的输入。

2-gram下Level-1抽象级别：

2-gram下Level-2抽象级别：

3-gram下Level-1抽象级别：

3-gram下Level-2抽象级别：

4-gram下Level-1抽象级别：

4-gram下Level-2抽象级别：

所以最终选择的参数如表x：

表x n-gram下不同抽象级别参数选择

|  |  |  |
| --- | --- | --- |
|  | Level-1 | Level-2 |
| 2-gram | mtry=;ntree= | mtry=;ntree= |
| 3-gram | mtry=;ntree= | mtry=;ntree= |
| 4-gram | mtry=;ntree= | mtry=;ntree= |

**3.4.2 SVM**

很多研究已经证明支持向量机（Support Vector Machine，SVM）是一种强大的分类工具，可以被广泛的应用与不同的领域。与随机森林算法不同，在SVM训练中，从输入数据到输出结果的过程并不清晰，也难以解释，因此SVM属于黑盒算法。

SVM通过定义核函数将输出数据映射到高维特征空间上，并在此空间中构造一个最有分类超平面（或者一组超平面），使得高维特征空间内的各个类的边缘间隔最大化。定义这些超平面的向量就被称为支持向量，如图x中被正方形圈出的o和x：



图x SVM分类示意图

SVM首先构建一个超平面能够最大化间隔距离，然后将定义扩展到非线性可分问题上，最后，将数据映射到一个高维空间，使得数据能够更容易地被线性边界分离开来。简而言之，SVM可以看作高维空间的线性算法。

SVM的优势在于利用了面向工程问题的核函数，能够提供准确率非常高的分类模型，同时借助正则项可以避免模型的过度适应，用户也不必担心诸如局部最优和多重共线性难题。SVM主要弊端是对模型进行训练和测试的速度比较慢，模型处理的时间冗长，因此算法不适合，应用于规模庞大的数据集。本文采用的数据集规模不是很庞大，因此可以使用SVM算法进行实验。另外，SVM的结果很难解释，如何确定核函数也是一个难点。

本文使用R语言中e1017包的libsvm函数训练支持向量机。通过训练函数svm，用户可以确定核函数、成本函数（即惩罚因子）和gamma函数。对于核函数，通过大量实验，本文使用效果最好的也是默认的radial（径向函数）作为核函数；对于gamma和惩罚因子，通过tune.svm来寻找最佳的gamma和惩罚因子。本文中，将gamma参数的可能范围设置为，惩罚因子选择1，10和100，使用tune.svm函数得到12组不同的参数组合。函数采用10遍交叉检验的方法获得每次组合的错误偏差，最后选择误差最低的最优参数组合。

**3.4.3 KNN**

K近邻（K-Nearest Neighbor，KNN）算法属于一种无参惰性学习方法学习。无参类算法不会对数据的分布做任何的假设，而惰性学习方法则不要求算法具备显性学习过程。

KNN算法采用相似性（距离）测量来训练所有样本并对新的样本进行分类标记。常用的相似性测量如下公式：  
欧氏距离：

曼哈顿距离：

KNN算法中，新到达的样本将被分配到某个在KNN中普通的类中。如果K=1，新样本将会被分配到距离其最近的类别中。算法仅要求用户输入K的值，如果K值比较小，有可能产生过度适应的问题。相反，如果给的K值过大，则有可能导致低度拟合。可以通过交叉验证的方式得到比较合适的K值。

KNN算法的优势：

1. 学习成本为0；
2. 无参意味着用户不需要对数据的分布类型做假设；
3. 只要对给定样本选取和式的距离测量方法，算法就可以处理任意类型的数据。

KNN算法的不足：

1. 算法难以理解；
2. 如果数据集比较大，算法计算代价非常高；
3. 算法性能依赖数据集维度的大小，如果要处理高维数据，应先对数据进行降维操作以提高算法的过程性能。

针对KNN算法不足2)，本文用于实验的数据，总恶意代码数目为10868，数据集并未大到KNN处理不了；对于不足3），本文事先对操作码做了抽象化处理，除了Level-0的3-gram和4-gram维度较大，其他的维度都维持在可接受的范围。

本文使用R语言class包中的knn函数来处理实验数据。

**3.5 本章小结**

本章主要介绍了基于机器学习算法的恶意代码检测技术。重点介绍了操作码抽象及特征分析的过程：首先，对数据查壳和脱壳处理，并对样本进行反汇编并得到汇编操作码序列；然后，对汇编操作码进行3种级别的抽象化，分别为Level-0、Level-1和Level-2，得到其对应的中间码；接着，分别对中间码进行n-gram算法提取特征，应用关联分析技术对提取到的n-gram词组进行统计分析，实验得出各个类别的频繁项集平均相似基本小于50%，因此本文使用概率矩阵作为机器学习算法的输入；最后，介绍了三种机器学习算法，以及本文如何应用这三种机器学习算法进行实验。

**第四章 系统设计与实验结果**

**4.1 系统设计**

下面详细介绍本文的系统框架。首先，对PE文件进行查壳与脱壳操作；其次，使用objdump工具对PE文件进行反汇编，得到汇编文本文件；再次，使用n-gram算法提取特征，并选择合适的抽象级别对特征进行映射操作，得到中间码的特征形式；然后，应用关联分析技术对中间码特征进行分析，并根据中间码的特征文本得到其对应的概率矩阵；最后，将概率矩阵作为机器学习算法的输入，训练模型并评估。系统框架如图x所示：



图x 系统框架图

本文系统在windows10系统下PowerShell+CodeBlocks+RStudio平台下，使用PowerShell脚本、C++语言和R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；其次，使用objdump工具对恶意代码进行反汇编，得到汇编代码文本；接着，在汇编文本中提取汇编操作码后，使用n-gram算法提取特征，并根据动态配置的映射关系对汇编操作码特征进行抽象处理，最终得到中间码版本的特征，为提高效率，本文使用了多线程的技术处理这一步骤；然后，使用关联分析技术对特征进行分析，使用频繁项集的平均相似度证明了概率矩阵的可行性，并构造概率矩阵；最后，将概率矩阵作为随机森林算法、SVM和KNN算法的输入构造模型，并使用caret算法包对模型调参及k折交叉验证。

**4.2 评价方法**

为了确保模型能对未知或新到达的对象进行正确预测，需要对模型性能进行评估，避免模型可能存在过拟合问题。

k折交叉验证技术能够解决过拟合问题，因此被广泛应用于分类器性能评估领域。k折交叉验证并不需要使用整个数据集，相反，它会将数据集划分为训练集和测试集两部分。这样，基于训练集得到的模型就可以通过测试集来完成性能测评。重复执行n次k折交叉验证后，就能够根据n次检验的平均正确率实现对模型的真实评估。

本文使用R语言caret算法包分别对随机森林算法、SVM和KNN算法进行10折交叉验证。

**4.3 实验数据**

本文使用的实验数据来自Kaggle上微软发起的一个恶意代码分类的比赛，使用的数据集的大小为136GB，其中样本种类分布如表x，

表x 恶意程序数据集

|  |  |
| --- | --- |
| 恶意程序家族 | 数量（单位，个） |
| Ramnit | 1541 |
| Lollipop | 2478 |
| Kelihos\_ver3 | 2942 |
| Vundo | 475 |
| Simda | 42 |
| Tracur | 751 |
| Kelihos\_ver1 | 398 |
| Obfuscator.ACY | 1228 |
| Gatak | 1013 |
| total | 10868 |

**4.4 实验结果对比**

**4.4.1 准确率比较**

**4.4.2 误报率比较**

**4.4.3 漏报率比较**

**4.4.4 实验结果分析**

**4.5 本章小结**