北京大学

硕士研究生学位论文

题目：**PDF 恶意代码检测技术研究与实现**

姓　　名：	孟正
学　　号：	1201210749
院　　系：	软件与微电子学院
专　　业：	软件工程
研究方向：	网络与系统安全
导师姓名：	文伟平　副教授

二〇一五年七月

版权声明

摘要

近年来，针对商业组织和政府机构的网络攻击事件层出不穷，APT 攻击时有发生。APT 攻击通常会采用窃取敏感信息、网络钓鱼、监控组织以及扰乱组织运行等方式实施破坏。恶意 PDF 文件是 APT 攻击的重要载体，它通过执行嵌入在文件内部的恶意代码完成攻击过程。由于恶意 PDF 文件具有严重危害性，因此，如何检测 PDF 文件中的恶意代码就成了计算机安全领域的研究热点。本文开展的研究工作对于保护用户隐私和改善网络安全现状具有重要意义。

本文以 PDF 文本检测和 PDF 漏洞检测作为主要研究内容，首先对 PDF 文件格式与常见的 PDF 攻击技术进行介绍，然后开展了以下工作：

1. 针对现有 PDF 恶意代码检测模型的不足，提出一种新的检测模型 PDFCheck，该模型包括基于主动学习策略的 PDF 文本检测子系统和 PDF 漏洞检测子系统，对 PDFCheck 模型的整体框架、检测流程和评价指标等进行描述。
2. 介绍基于主动学习策略的 PDF 文本检测子系统，对 PDF 文件解析、Javascript 攻击检测、元数据和文件结构特征提取、特征选择以及分类等关键技术进行探讨，分析基于本地 Hook 技术的 Javascript 代码提取方法，研究基于主动学习的 SVM 算法在 PDF 文本分类中的应用，并将该算法与基于监督学习的 SVM 算法进行比较，实验结果表明，基于主动学习的 SVM 算法具有更好的分类性能。
3. 对 PDF 漏洞检测子系统进行描述，介绍 PDF 漏洞分析与检测规则库构建的流程，提出一种基于规则匹配的 PDF 已知漏洞检测方法，并对堆栈检测、函数调用检测、内存检测和动态链接库检测四种 ROP 链检测方法进行分析。
4. 使用 Visual Studio 2010 集成开发环境对 PDFCheck 的关键模块予以实现，并展示其运行结果。将 PDFCheck 与现有的 PDF 恶意代码检测模型和漏洞检测工具进行比较，实验结果表明，PDFCheck 检测准确率高、检测方法多样且检测能力强，效果更为突出。

关键词：PDF 恶意代码，文本检测，主动学习策略，漏洞检测，特征匹配

Research and Implementation of PDF Malicious Code Detection Technology

Meng Zheng(Software Engineering)
Directed by Wen Weiping

ABSTRACT

In recent years, cyber-attacks against organizations and businesses have increased, and APT attacks have occurred from time to time. Attacks aimed at organizations usually take the form of harmful activities such as stealing confidential information, spying and monitoring an organization, and disrupting an organization's actions. The malicious PDF file is an important carrier for cyber-attacks and it completes the attack process by executing the malicious code embedded in the file. For the problems discussed above, how to detect malicious code in the PDF file has been a hotspot in the field of computer security. The research is of great significance for protecting users' privacy and improving the network security situation.

The main research contents of this paper focus on PDF texts detection and PDF vulnerabilities detection. First of all, this paper introduces the PDF file format and the common PDF attack technologies. Then we carry out the following work:

1. For the deficiencies of the existing PDF malware detection model, a new detection model PDFCheck was proposed. PDFCheck includes PDF text detection subsystem based on active learning strategy and PDF vulnerabilities detection subsystem. The overall framework, detection process and evaluation indicator of PDFCheck were described.
2. The PDF text detection subsystem based on active learning strategy was introduced and the key technologies such as PDF file parser, Javascript attack detection, metadata and file structure feature extraction, feature selection and classification was investigated. Next, the Javascript code extraction method based on hooking local JS engine was analyzed. The application of SVM algorithm based on active learning in PDF text classification was studied and comparing this algorithm with SVM algorithm based on supervision learning. The result shows that the SVM algorithm based on active learning has better performance.
3. The PDF vulnerabilities detection subsystem was described and the process of

analyzing the vulnerability principle, constructing detection rule was introduced. A detection method of PDF known vulnerabilities based on rule matching was proposed and the four kinds of detecting methods of ROP chain, stack detection, function call detection, memory detection and dynamic link library detection was analyzed.

4. Implementing the key modules of PDFCheck by using Visual Studio 2010 integrated development envirionment and showing their interface. Comparing PDFCheck with existing PDF malicious code detection model and existing vulnerability detection tools and the results show that PDFCheck has higher detection accuracy, more detection methods and stronger detection ability, which has prominent effect.

KEY WORDS: PDF malicious code, Text detection, Active learning strategy, Vulnerability detection, Feature matching

目录

第一章　绪论

1.1　研究背景和意义

从 2009 年开始，针对商业组织和政府机构的网络攻击事件数量急剧攀升。据卡巴斯基实验室统计，2013 年约有 91%的组织机构遭受网络攻击[1]。这些攻击通常会采用窃取敏感信息、网络钓鱼、监控组织以及扰乱组织运行等方式实施破坏。电子邮件（例如 email）是一种在发送者和接收者之间传递数字信息的有效方法，目前已广泛地被各大组织结构所采用。因此，包含恶意代码的电子邮件附件就成为攻击者进行网络攻击的有效手段。2014 年 1 月 15 日发生的一起针对以色列国防部的网络攻击事件就是通过电子邮件附件实施的，当附件中的恶意 PDF 文件被打开后，PDF 文件会释放木马，从而使计算机被攻击者控制。

对于一个组织机构的网络安全团队来说，其主要职责是防止攻击者对内部网络进行渗透攻击和实施破坏，因此，他们部署了入侵检测系统[2]、入侵防御系统[3]、防病毒软件和防火墙等安全防护工具。但是，这些工具不能检测和识别包含在电子邮件内部的攻击，尤其是不能对复杂的 APT（Advanced Persistent Threat，高级持续性威胁）攻击进行有效抵御。

为了使接收者打开恶意邮件、恶意附件或点击非法链接，攻击者通常使用社交工程[4]的手段。例如，攻击者可以发送一个具有醒目标题和复杂正文的邮件，从而吸引接收者打开附件。网络安全公司 Trend Micro 经调查发现[5]，大多数针对政府部门和大型公司的 APT 攻击都是依赖于网络钓鱼的。

由于大多数邮件服务器可以阻止邮件附件中的可执行文件被传递，因此，在最新的网络攻击中，不可执行的文件发挥了越来越重要的作用。这些文件有一定的格式，它们仅可以被特定程序读取，而不能直接运行。以 PDF 文件为例，它只能被 Adobe Reader 或 Foxit Reader 打开。人们普遍认为，不可执行文件的安全性高于可执行文件，因此，他们拒绝打开经电子邮件传递的可执行文件，却轻易打开了自认为安全的 Microsoft Office 或 PDF 文档，致使隐藏在其中的恶意代码执行，从而引发网络攻击。F-Secure 公司的报告[6]显示，2010 年第一季度，Adobe Reader 相关的攻击占全部文档类攻击的 61%，而与 Microsoft Office 相关的攻击只占据 24%，如图 1.1 所示。

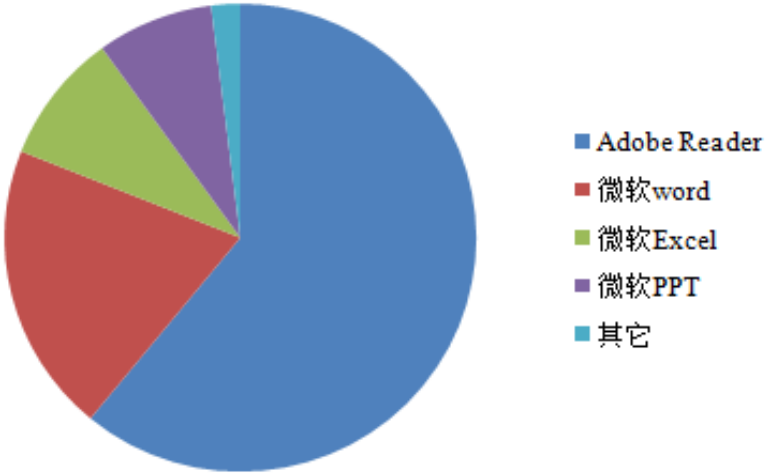

图 1.1　文档攻击分布图

本文对 PDF 恶意代码检测的关键技术进行研究和分析，提出一套完整的检测方案，并完成工具的设计与实现。本文开展的研究工作对于识别、检测恶意 PDF 文件，保护用户隐私和改善网络安全现状具有重要意义。

1.2　国内外研究现状

当前，对 PDF 恶意代码进行检测主要有两种方法：静态分析方法与动态分析方法。静态分析方法对文件或应用程序的代码进行分析，不需要真正执行它。而动态分析方法则要执行文件或应用程序，并在其运行时监控它们的状态和行为。近年来，安全研究人员对 PDF 恶意代码检测技术进行了一些研究，并取得大量的研究成果。其中大多数采用了静态分析方法，这是因为静态分析需要更少的计算资源，运行速度快。静态分析方法通常检查嵌入在 PDF 文件中的 Javascript 代码[7]或元数据（包括 PDF 文件的对象、结构、特定流的数量以及关键字等）。但是，静态分析方法也有其局限性，它不能检测混淆后的代码，这些代码在运行时会表现出恶意行为，而动态分析方法能够对混淆后的恶意 PDF 文件进行有效检测。

图 1.2 描述了安全研究人员在 PDF 恶意代码检测方面所开展的一些研究工作[8]。

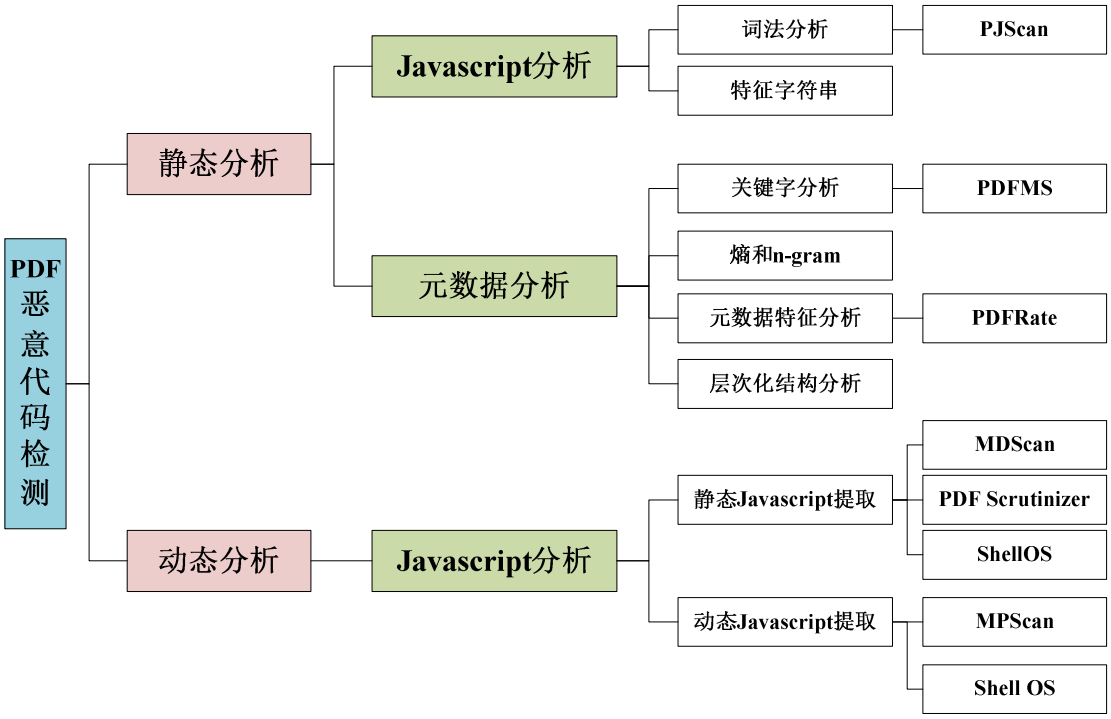

图 1.2　PDF 恶意代码检测相关研究

1.2.1　基于静态分析的检测方法

基于静态分析的检测方法包括 Javascript 分析和元数据分析两方面内容，该方法在不执行或不打开 PDF 文件的情况下对 PDF 文件的特定内部组件进行分析。Javascript 代码是一个重要的 PDF 组件，大多数利用 PDF 文件实施的攻击都与内嵌在其中的 Javascript 代码有关。Laskov 和 Srndic[9]、Vatamanu[7]等人通过分析 Javascript 代码对恶意 PDF 代码进行检测。PDF 文件的元数据包括与文件内容有关的元特征，相比于 Javascript 代码，PDF 文件中的元数据不会被代码混淆技术影响，Srndic 和 Laskov[10]，Smutz 和 Stavrou[11]，Maiorca[12]等人提出了一些针对元数据的静态分析方法。

对于静态分析方法来说，PDF 文件解析至关重要，解析器应能够对文件中的全部信息进行提取，包括加密或混淆的数据。如果重要的 PDF 文件信息（如 Javascript 代码）被遗漏，那么将导致恶意的 PDF 文件被误报为正常文件。

由图 1.2 可知，Javascript 分析可采用词法分析和特征字符串识别两种技术，它们将从 PDF 文件中提取的 Javascript 代码序列化，通过机器学习算法建立分类模型。PJScan[13]是一种针对 PDF 文件的静态检测模型，该模型是由 Nedim Srndic 和 Pavel Laskov 于 2012 年研发的，它采用 Javascript 词法分析技术，针对嵌入的 Javascript 代码进行特征提取，根据 OCSVM（One-Class Support Vector Machine，单类支持向量机）进行分类。

元数据分析的常见方法有关键字分析、层次化结构分析、元数据特征分析以及词

频和熵分析等。其中，Maiorca[12]等人分析了嵌入关键字的频率，Srndic 和 Laskov[10]分析了层次化结构路径，Pareek[14]等人对整个 PDF 文件中字符序列的熵集合进行计算，Smutz 和 Stavrou[11]使用了特定的元数据特征。这些方法都使用了 PDF 文件中对象或结构的统计信息，而不是 PDF 文件的文本或代码。

1.2.2 基于动态分析的检测方法

基于动态分析的检测方法可以动态执行嵌入在 PDF 文件中的 Javascript 代码，该方法在特征提取或分析阶段应至少包含一个动态的步骤。由于提取 Javascript 代码的方式不同，基于动态分析的检测方法又可以分为两大类：静态 Javascript 提取和动态 Javascript 提取。MDScan[15]、Scrutinizer[16]采用静态 Javascript 提取技术，它们从一个 PDF 文件中静态提取嵌入在其中的 Javascript 代码，然后通过 Javascript 引擎执行代码，并在代码运行过程中，对可疑或恶意的行为进行监控。MPScan[17]采用动态 Javascript 提取技术，它在运行时动态地提取 Javascript 代码。ShellOS V2[18]也采用了基于动态分析的检测方法，它是一个通过硬件虚拟化技术实现的轻量级操作系统，用以执行代码流，ShellOS 只有约 2500 行代码，它以宿主机的形式在 KVM 上运行。

一般来说，Javascript 提取过程耗费的系统资源越多，动态分析方法的检测能力就越好。然而，有时候采用静态 Javascript 提取技术会导致提取到的 Javascript 代码不能表示文件的真正行为。例如，代码被混淆或代码被放置于 PDF 文件中一处不规则的位置。Javascript 代码的动态提取技术能够克服这些弱点。

另外，对整个 PDF 文件进行动态分析需要消耗较多的系统资源，因此，安全研究人员仅分析了其中的 Javascript 代码。

1.3 论文的研究内容

为有效保护用户隐私，提升网络安全形势，本文对 PDF 恶意代码检测的关键技术进行研究，并完成检测模型的设计与实现。本文开展的研究工作主要有下述五个方面：

1）分析 PDF 文件格式，对常见的 PDF 攻击技术进行分类和归纳

分析 PDF 文件的逻辑结构和物理结构，研究攻击者利用 PDF 恶意代码实施网络攻击的方法及特征。

2）提出新的 PDF 恶意代码检测模型

从产生时间、检测方法和检测能力三个方面对现有的 PDF 恶意代码检测模型进行评价，分析它们存在的不足，并以此为基础，提出一种新的检测模型。

3）PDF 文本检测关键技术

研究 PDF 文本检测的关键技术，包括 PDF 文件解析、Javascript 攻击检测、元数

据和文件结构特征提取以及 SVM 分类等。研究 Hook Adobe Reader 本地 JS 引擎的方法在 Javascript 代码提取上的应用，研究基于主动学习策略的 SVM 分类算法的基本原理和实现流程，并将该算法与基于监督学习的 SVM 分类算法进行比较。

4）PDF 漏洞检测关键技术

对 PDF 漏洞的成因进行分析，编写漏洞检测规则，建立 PDF 已知漏洞特征库，研究规则匹配方法在 PDF 已知漏洞检测中的应用；另外，对 PDF 恶意文件中的漏洞利用关键代码，如 ROP（Return-oriented programming，面向返回的编程）链等进行检测。

5）PDF 恶意代码检测模型的实现与测试

以上述研究为基础，对本文提出的 PDF 恶意代码检测模型进行编码实现，并将该模型与现有的其它 PDF 恶意代码检测模型进行比较。

1.4 论文的组织结构

第一章：绪论。简单介绍课题的研究背景，描述国内外在 PDF 恶意代码检测方面的研究工作，分析课题的研究内容和论文的组织结构。

第二章：PDF 文件格式与攻击技术简介。首先对 PDF 文件的逻辑结构和物理结构进行详细描述，然后介绍 PDF 文件的常见攻击技术，分析 PDF 恶意代码的实现原理和特征。

第三章：PDF 恶意代码检测模型 PDFCheck。首先分析现有检测模型的缺点，并针对其不足提出一种新的 PDF 恶意代码检测模型 PDFCheck，对模型的整体框架和主要流程进行说明，然后介绍 PDF 恶意代码检测模型的评价指标、测试数据和实验环境。

第四章：基于主动学习策略的 PDF 文本检测子系统。首先介绍子系统的总体架构，然后依次分析 PDF 文件解析、Javascript 攻击检测、特征提取、特征选择和分类等关键模块，通过实验确定特征数量和 SVM 算法的最佳参数 C，比较基于主动学习的 SVM 算法与基于监督学习的 SVM 算法的测试结果。

第五章：PDF 漏洞检测子系统。首先对 PDF 漏洞检测子系统的总体架构进行介绍，然后结合 PDF 漏洞分析实例，对漏洞检测规则库进行构建，提出一种基于规则匹配的 PDF 已知漏洞检测方法，接下来描述 ROP 技术的原理，分析检测 ROP 链的若干方法。

第六章：PDFCheck 关键模块实现与测试。首先介绍 PDFCheck 的开发环境，然后对基于主动学习策略的 PDF 文本检测子系统和 PDF 漏洞检测子系统分别予以实现，展示其运行结果。从检测精确度、检测特定攻击能力等方面将本文提出的模型 PDFCheck 与现有的其它模型进行比较，并比较 PDFCheck 与安全检测工具赛门铁克、BitDefender 的已知漏洞检测能力。

第七章：总结与展望。对本文所做的主要工作予以总结，提出后续的研究方向。

第二章　PDF 文件格式与攻击技术简介

2.1　PDF 文件格式

PDF（Portable Document Format），又被称作便携式文档，是一种被广泛使用的，用于电子文档分发的开放式标准，它最早是被 John Warnock 构思出来的[19]。PDF 文件不仅是简单的文本，它还可以包含图像与其它多媒体元素，PDF 文件可以设置密码保护，也可以执行 Javascript 代码。当前，PDF 文件已经被所有主流的 PC 操作系统和移动平台所支持，例如 Windows，Linux，Mac OS，Android，Windows Phone 以及 iOS[20]。

2.1.1　PDF文件逻辑结构

PDF 文件在逻辑上呈现出树形组织结构，其形状如图 2.1 所示。由图可知，PDF 文件的根对象 Catalog 是由树的根节点来表示的，它包含 Pages（页面组）、Outlines（书签）、URI（统一资源标识符）、Metadata（元数据）等字段[21]。

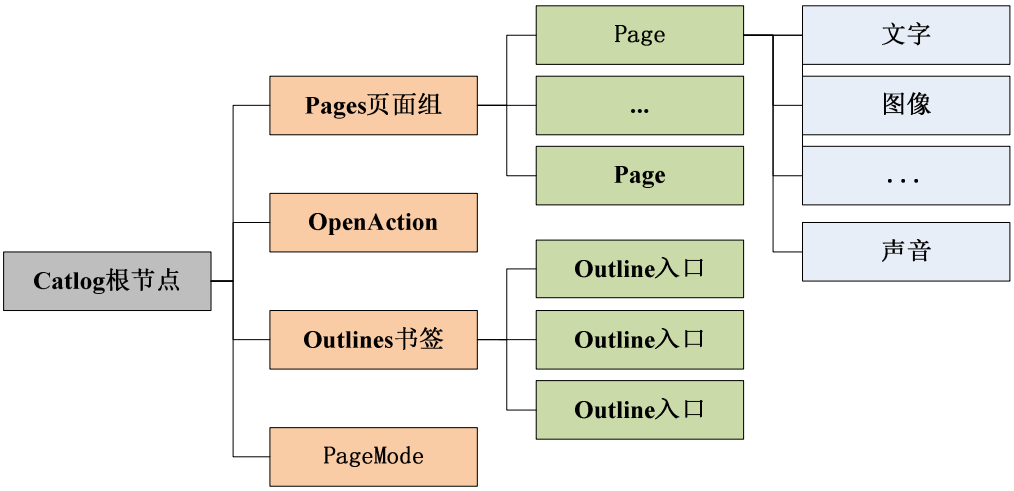

图 2.1　PDF 文件的逻辑结构

PDF 文件逻辑结构的核心就是 Catalog 根节点，它是连接 PDF 文件逻辑结构和物理结构的桥梁，是解析文件的起始点。由于 Catalog 根节点包含的内容种类繁多，下面仅对最为重要的几个字段进行简要说明。

1）Pages 字段

Pages 是每个 PDF 文件都必须包含的字段，用来表示 PDF 文件中所有页面的集合。Pages 字段属于 Dictionary 类型，它包含几个重要的子字段，如表 2.1 所示：

表 2.1　Pages 中字段的含义

字段	类型	值
Type	name	（必选字段）只能是 pages
Parent	dictionary	当前节点的父节点
Kids	array	一个数组，它是由间接对象构成的，其节点是 page tree 或 page
Count	integer	Page tree 中叶子节点的个数

由表 2.1 可知，Pages 字段最重要的作用就是对所有的 Page 对象进行管理。Page 对象保存着 PDF 文件的页面属性（Property）、元数据（MetaData）、注释（Annots）等信息。

2）Outlines 字段

Outline，又被称为书签，该字段建立了页面位置和书签名之间的对应关系，用户可以通过书签对指定页面的内容进行阅读，从而方便用户由 PDF 文件的一部分跳转至另一部分。书签呈现树状结构，它可以将 PDF 文件的结构直观地展示给用户。Outlines 主要包括以下几个字段，如表 2.2 所示：

表 2.2　Outlines 中字段的含义

字段	类型	值
Type	name	如果该字段不空，那么其值必须为 Outlines
First	dictionary	第一个顶层 Outline item
Last	dictionary	最后一个顶层 Outline item
Count	integer	Outline 所有层次的 item 总数

由表 2.2 可知，Outlines 对象是一个 Outline item 管理器，而 Outline item 则表示文字、动作、图像和目标区域等。

3）URI 字段

URI（Uniform Resource Identifier，统一资源标识符）对 PDF 文件中的链接信息进行定义。PDF 文件可以通过该字段管理文件和目录中的链接。

4）Metadata 字段

Metadata 字段是以 XML 文件的方式呈现的，它表示 PDF 文件中的一些附带信息，这就使我们在不解析整个 PDF 文件的情况下就获得文件的大致信息。

5）其它字段

在 Catalog 字典中，还有其它一些较为常用的字段，如 Version 字段、OpenAction 字段、Dests 字段、PageLabels 字段、Names 字段、Threads 字段、PageLayout 字段、

PageMode 字段、OpenAction 字段、AA 字段等。

2.1.2　PDF文件物理结构

1）PDF 对象

对象是 PDF 文件的基本元素，包括间接对象与直接对象两种，其中，间接对象是通过一个数字被引用的，直接对象不是通过数字被引用的。

PDF 对象有 Boolean、String、Numeric、Name、Array、Null、Dictionary 和 Stream 等八种类型，每种类型的含义如表 2.3 所示：

表 2.3　PDF 对象类型及其含义

PDF 对象类型	含义
Boolean	布尔数据类型，可以作为 dictionary 对象的条目或 array 对象的元素
Numeric	分为整型和实型两种，支持十进制数字，不对指数形式的数字提供支持
String	可分为十六进制字串和直接字串两种，其中，十六进制字串是包含在“<>”中的十六进制数字，直接字串是包含在“()”中的字符序列
Name	以“\”开头的 8 位字符序列
Null	表示为空
Array	包含在“[]”中的有序对象序列，“[]”中的内容可以是任何对象类型，包括 Array
Dictionary	包含在“<< ”和“>>”中的若干条目，每个条目是由 key 与 value 两部分组成的，其中，key 具有唯一性，PDF 文件中的大部分间接对象是 dictionary 类型的
Stream	是一个特殊的 dictionary 对象，包含于关键字“stream”和“endstream”之间。stream 对象用于存储图片、脚本代码和文本等数据。stream 中的信息有可能进行压缩或加密处理，stream 之前的 dictionary 包含了是否解密和如何解密 stream 的相关信息

2）PDF 文件结构

PDF 文件结构定义了对象被访问和被更新的方法，一个合法的 PDF 文件由以下四部分构成：文件头（Header）、文件体（Body）、交叉引用表（Cross reference Table）和 Trailer，下面我们逐一进行介绍。

①文件头

文件头位于 PDF 文件的第一行，它对 PDF 文件的版本号进行标识，文件头的格式

是：%PDF-[version number]。

②文件体

文件体是 PDF 文件的主要组成部分，它包含所有 PDF 对象。在对象结构中，我们可以对文本/字体/图像/书签/视频/超链接等进行定义。文件体的格式如图 2.2 所示。

```
3 0 obj
...
endobj
```

图 2.2　文件体格式

在图 2.2 中，第一行的“3”表示对象序号，该序号唯一标识一个对象；“0”表示该 PDF 文件被创建之后没有发生改变，它一旦被修改，“0”就开始累加；“obj”和“endobj”用来定义对象的范围；省略号则表示 PDF 文件中的对象。

③交叉引用表

交叉引用表对内存中每一个间接对象的起止位置进行包含，它支持对 PDF 文件对象的随机访问。在交叉引用表中，每一个对象都有一个入口，其长度通常是 20 字节。交叉引用表的格式如图 2.3 所示。

```
xref
0 1
0000000000 65535 f
3 1
0000025325 00000 n
23 2
0000025518 00002 n
0000025635 00000 n
30 1
0000025777 00000 n
```

图 2.3　交叉引用表格式

xref 表示交叉引用表的开始，每个交叉引用表包含多个子段，各个子段的第一行都有两个数字，其中，数字一表示对象起始号，数字二表示连续对象的个数。接下来是每个对象的具体信息，其中，第一个数字段表示该对象相对于文件头的偏移地址，第二个数字段用于标记 PDF 文件的更新信息，最后一位 n 或 f 表示是否使用对象（n 表示使用对象，f 表示对象被删除或没有使用对象）。图 2.3 中的交叉引用表共包含 4 个子段，每个子段分别拥有 1 个、1 个、2 个和 1 个对象，第一个子段的对象不可用，其余子段对象皆可用。

④Trailer

PDF 阅读器在对文件进行解析时，可以通过 trailer 快速定位交叉引用表，从而找到各个对象的位置，还可以通过 trailer 中的 dictionary 获得 PDF 文件的全局信息，如关键字、作者、标题和加密信息等。图 2.4 描述了 trailer 的格式，其中，%%EOF 表示文件结尾，之前的两行是文件偏移和关键字 startxref，关键字 trailer 之后是一个 dictionary，它包含以下组成部分：Size（Integer）、Info（dictionary）、Root（dictionary）和 ID（array）。

```
trailer
<< /Size 22
/Root 2 0 R
/Info 1 0 R
/ID [ <81b14aafa313db63dbd6f981e49f94f4 >
<81b14aafa313db63dbd6f981e49f94f4 >
]
>>
startxref
18799
%%EOF
```

图 2.4　trailer 格式

图 2.5 描述了 PDF 文件的整体结构[8]。

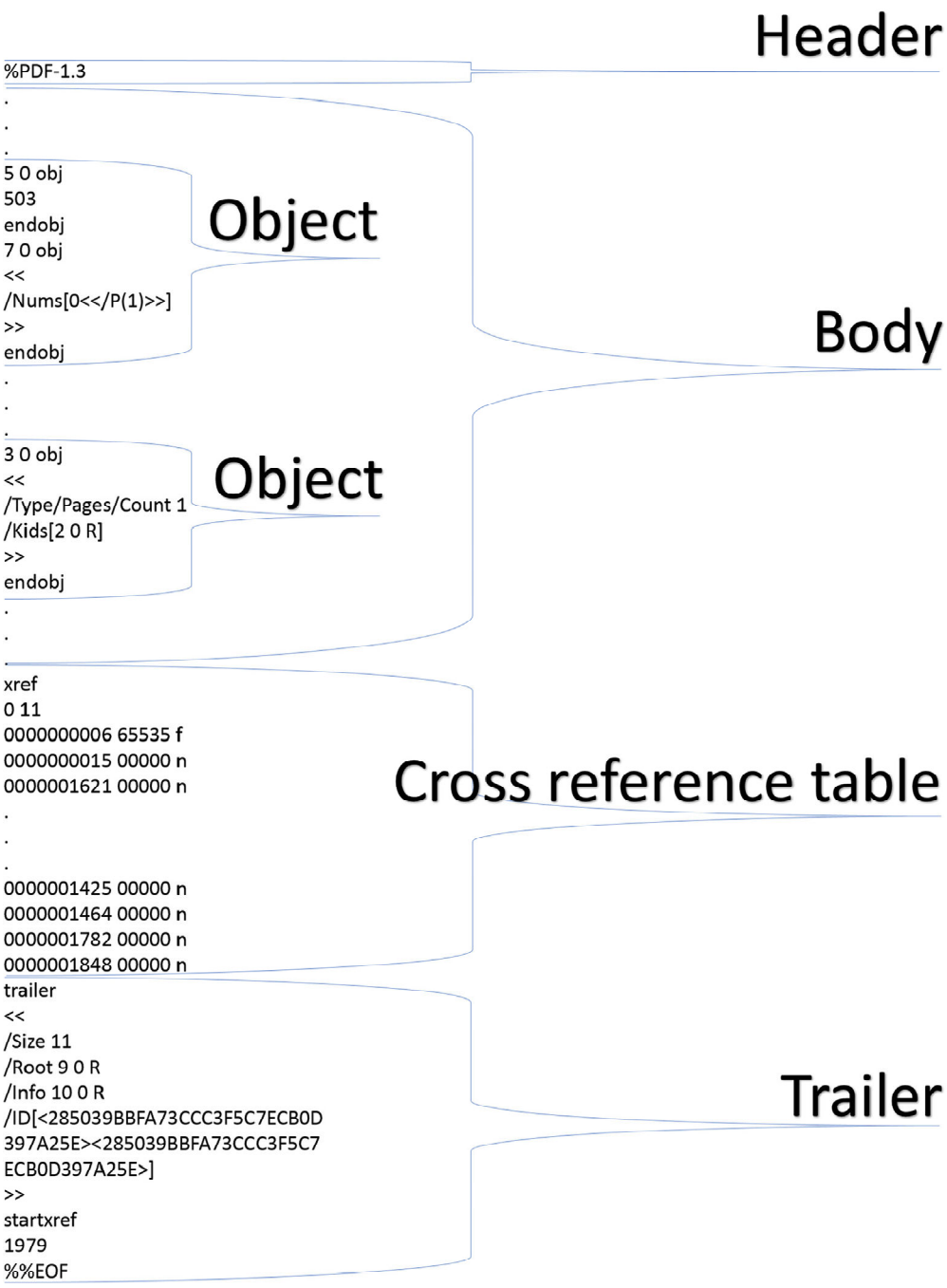

图 2.5　PDF 文件结构

2.2　PDF 攻击技术

2011 年发布的 Adobe Reader Version X 包含新特性 PMAR（Protected Mode Adobe Reader）。当 Adobe Reader 读取 PDF 文件时，PMAR 能够使用沙盒[22]技术创建一个隔离的环境，从而使 PDF 恶意代码的行为不能对操作系统产生影响。但是，大多数组织结构并没有对 PDF 阅读器进行更新，它们仍在使用旧版本的 Adobe Reader。另外，通

过一些技术手段，PDF 恶意代码可以绕过沙盒，实施攻击。下面对 PDF 文件的常见攻击方式进行介绍。

2.2.1　Javascript代码攻击

PDF 文件中包含的 Javascript 代码可以实现一些合法功能，如 3D 显示、表单验证等。Javascript 代码可以被放置于本地主机或远程服务器上，可以通过 URI 链接获取 Javascript 代码。判断 PDF 文件中是否嵌入 Javascript 代码的主要方法就是寻找 dictionary 中的关键字“JS”。但是当这个包含 dictionary 的对象被混淆后，那么“JS”关键字[12]在明文文本中也是不可见的，从而使依赖于关键字的静态分析方法失效。

在 PDF 文件中，恶意 Javascript 代码的主要目的是利用 PDF 阅读器（如 Adobe Reader）的安全漏洞，将程序的执行流程转移到嵌入的恶意 Javascript 代码上。这可以通过堆喷射技术[23]实现，堆喷射技术需要使用大量填充有 shellcode 的堆，通过特意构造，可使虚函数表的函数指针跳转入指定的地址空间。攻击者在 shellcode 之前会增加大量的 NOP 指令，从而提高 shellcode 执行的可能性。当函数指针跳转到 NOP 指令时，shellcode 便会随后被执行。堆喷射攻击的执行流程如图 2.6 所示。Javascript 代码的另一种恶意行为是从 Internet 上下载可执行程序，该可执行程序一旦执行，就会对用户主机进行攻击。通过 Javascript 代码也可以打开恶意网站，实施其它类型的攻击。

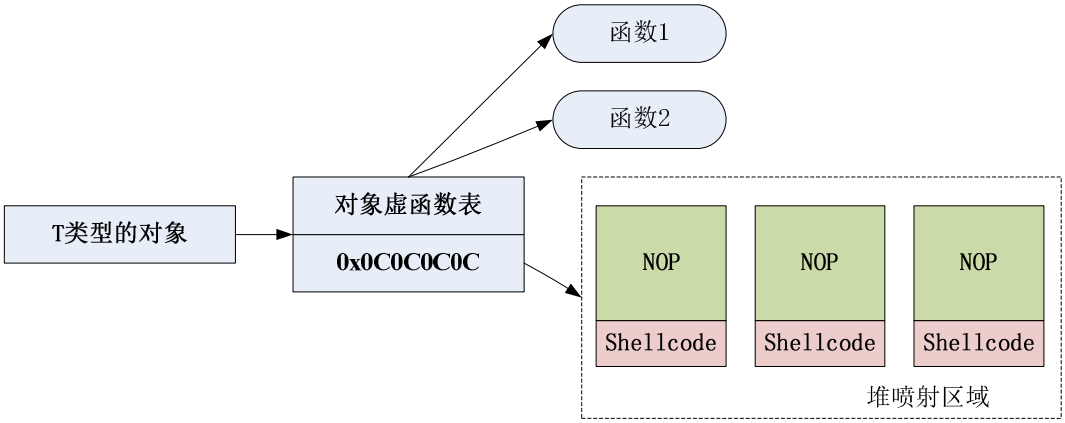

图 2.6　堆喷射攻击执行流程

代码混淆技术本用于防止应用程序被逆向和篡改，它也可以被攻击者使用去隐藏恶意的 Javascript 代码。混淆后的代码将难以被检测和阅读。表 2.4 对一些常被攻击者使用的 Javascript 代码混淆技术[24]进行简述。

表 2.4　Javascript 代码混淆技术

混淆技术	描述
分隔恶意代码	将恶意代码片段分布于多个对象中，在程序运行时，对其进行收集、合并和编译
添加随机空格	在恶意代码中添加随机空格，从而绕过基于特征码识别的恶意代码检测工具。由于 Javascript 代码忽略空格，不影响代码执行
添加随机注释	在恶意代码中添加随机注释，从而绕过基于特征码识别的恶意代码检测工具。由于 Javascript 代码忽略注释，不影响代码执行
变量名随机化	随机改变变量名，以绕过基于特征码识别的恶意代码检测工具
整型混淆	以其他方式表示数字，这种方法可被用于隐藏特定的内存地址
字符串混淆	对代码中的字符串进行替换，使其难以被阅读和分析，例如可以把一个字符串划分为多个子串
函数名混淆	将能够体现代码意图的函数名进行隐藏，可以创建一个具有随机名字的指向目的函数的指针
块随机	改变代码的语法而不是它的行为

2.2.2　嵌入文件攻击

一个 PDF 文件中可以嵌入其它类型的文件，例如 HTML、SWF、XLSX、EXE 和 Microsoft Office 等。攻击者如果要在正常文件中嵌入恶意文件，可以采用这种方式，这样，攻击者就能够利用嵌入在 PDF 文件中其它类型文件的漏洞执行恶意行为。通过一些技术手段，嵌入的文件能在用户不知情的情况下被打开。通常，为逃避检测，嵌入的恶意文件是被混淆的，另外 Adobe Reader 内置的黑名单也不允许 PDF 文件中嵌入的可执行文件被启动，但 Python 代码（*.py）并没有在黑名单里。

2013 年，Maiorca[25]等人提出了一种新型的攻击技术—反向模拟（Revrese Mimicry），这种技术是用来规避恶意代码检测工具的，它能够改变一个恶意文件的结构和对象，使其与正常文件相似。通过反向模拟技术，我们可以在 PDF 文件结构不发生改变的情况下向其中注入恶意内容。这种方法实现简单，不需要对恶意代码检测工具特征库中的 PDF 文件结构特征进行了解。

Maiorca 等人提出了三种实现反向模拟技术的方法：1）在一个正常的 PDF 文件中嵌入恶意的 EXE 文件；2）在一个正常的 PDF 文件中嵌入恶意的 PDF 文件；3）在 PDF 文件的非特定位置注入 Javascript 代码，使该文件不包含对其它对象的引用。通过基于监督学习算法的 PDF 恶意代码检测工具 PJScan[9]、PDFRate[11]和 PDFMS[12]对由第 3）种方法生成的 PDF 恶意样本进行检测，检测结果表明仅 PJScan 能够对这种攻击方式进行有效检测。

2.2.3　表单提交和URI攻击

2013 年，Valentin Hamon[26]提出了一种基于表单提交的攻击技术，攻击者可以使用该技术在一个 PDF 文件中执行恶意代码。Adobe Reader 允许用户通过 SubmitForm 命令由客户端向特定服务器提交 PDF 表单。使用 Wireshark 工具对 PDF 文件的提交表单操作进行捕获，如图 2.7 所示。

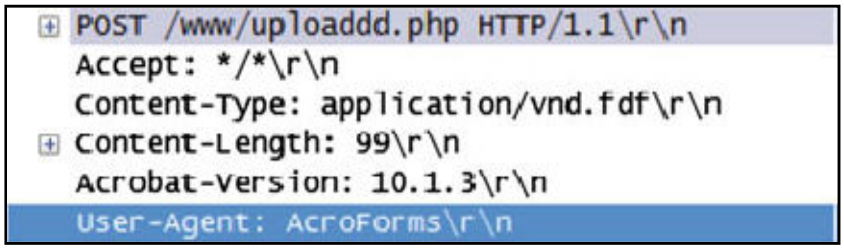

图 2.7　Wireshark 捕获的 PDF 提交表单操作

提交表单的文件格式有多种，其中一个是 FDF（Forms Data Format，表单数据格式），它是基于 XML 格式的，FDF 的文件结构及简单示例如图 2.8 所示。为了向特定 URL 发送数据，Adobe Reader 会生成一个 FDF 文件，如果此 URL 属于远程服务器，那么服务器将产生应答，应答数据会暂时存储在%APPData%目录中，而%APPData%是在 Web 浏览器上自动产生的。攻击者可以向恶意网站发送一个简单请求，此时，%APPData%目录会自动产生，从而使用户的 Web 浏览器被弹出，恶意网站可以利用用户的 Web 浏览器漏洞实施攻击。

另外，攻击者可以利用指向远端任何类型文件的 URI 地址，这些文件可以是可执行文件，也可以是不可执行文件，包括 exe 文件和 PDF 文件。下面描述一个攻击场景：当一个正常 PDF 文件向攻击者的 PHP Web 服务器提交表单后，服务器会通过正则表达式对表单头部的 Adobe Reader 版本信息进行判断。当服务器识别了用户的 Adobe Reader 版本之后，它将向用户发送一个恶意 PDF 文件，该文件利用了特定版本 Adobe Reader 的漏洞。

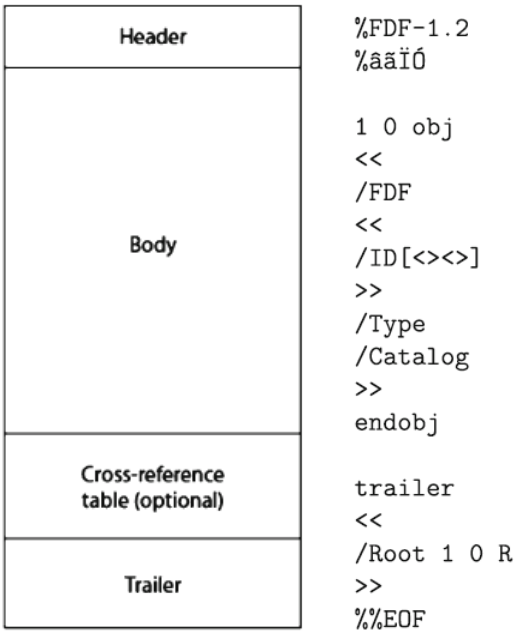

图 2.8　FDF 文件结构及简单示例

2.2.4　PMAR保护机制绕过

自从 Adobe Reader 引入 PMAR 保护机制以来，针对它的网络攻击数量急剧减少，PMAR 在一定程度上提升了攻击门槛，但是这并不意味着无法对其进行攻击。

一种攻击方法是修改注册表键值，关闭 Adobe Reader X 的沙盒机制。可以通过下述攻击场景实现：当用户打开 PDF 文件时，PDF 文件通过 Web 浏览器（URI 地址）自动地下载一个恶意的可执行文件，该可执行文件具有修改注册表键值的功能，从而使 Adobe Reader X 的 PMAR 保护机制和 Internet Explorer 的 URL 安全区域管理策略失效。

另一种攻击方法是利用 Adobe Reader 本身的安全漏洞。Adode Reader 的 XFA（XML Forms Architecture，XML 表单框架）处理模块的 AcroForm.api 中存在 CVE-2013-0640[27]漏洞，通过该漏洞可以获取任意代码的执行权限。在 Adobe sandbox 的 broker 进程中存在 CVE-2013-0641[28]漏洞，通过该漏洞可以实现从 sandbox 中逃逸的目的，从而获得最高权限。利用上述两个漏洞可以完成整个攻击过程，首先通过 CVE-2013-0640 漏洞对 AcroForm.api 的基地址进行泄露，然后利用这个基地址构造 ROP 链，从而突破 DEP（Data Execution Prevention，数据执行保护）防护机制，接下来利用 CVE-2013-0641 漏洞从沙盒中逃逸，最终使 Adobe 的 PMAR 防护机制失效。利用 Adobe Reader 本身的安全漏洞 CVE-2013-0640 和 CVE-2013-0641 突破 PMAR 的运行效果如图 2.9 所示，由图可知，当打开 PDF 文件后，计算器随即被弹出。

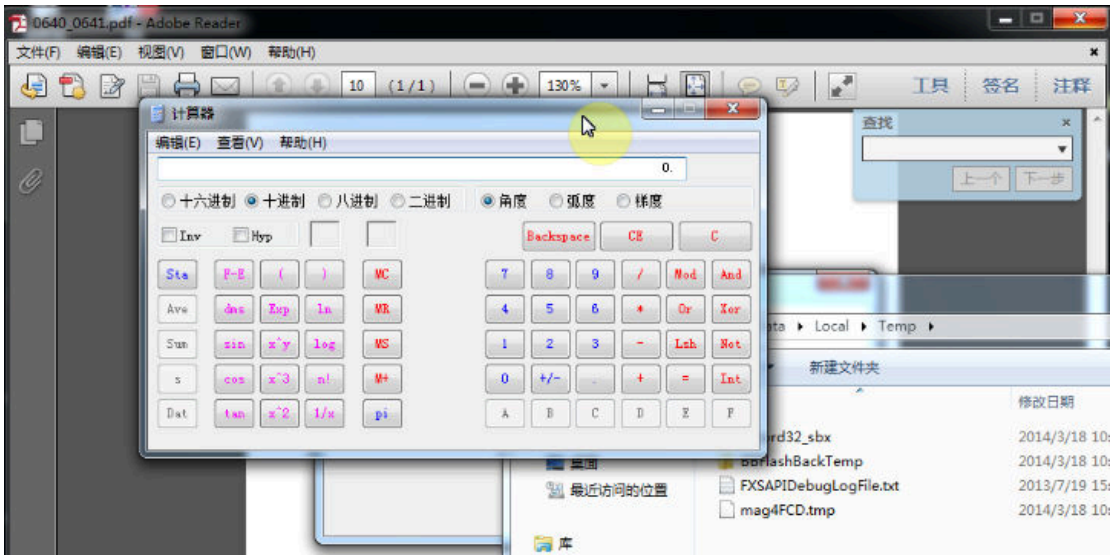

图 2.9　利用 Adobe Reader 本身漏洞突破 PMAR 的运行效果

2.3 本章小结

本章首先对 PDF 文件的逻辑结构和物理结构进行详细描述，然后介绍 Javascript 代码攻击、嵌入文件攻击以及表单提交和 URI 攻击等常见的 PDF 攻击技术，分析 PDF 恶意文件的实现原理和特征。

第三章　PDF 恶意代码检测模型 PDFCheck

3.1　现有模型的不足

由上文可知，当前主流的 PDF 恶意代码检测方法有静态分析和动态分析两种，针对每种检测方法，均形成了一些成熟的检测模型。现有的 PDF 恶意代码检测模型主要有 PJScan、ShellOS、MDScan、MPScan、PDFMS、PDF Scrunizer 等，现从产生时间、检测方法和检测能力三个方面对上述模型进行比较[8]。为清晰、客观地比较各种模型的检测能力，我们选取本文第二章描述的 PDF 常见攻击技术，包括恶意 Javascript 代码、代码混淆、反向模拟、远程加载恶意代码、恶意 URI 解析等，分析各种模型是否可以对上述攻击技术进行有效检测，比较结果如表 3.1 所示。

表 3.1　现有的 PDF 恶意代码检测模型比较

模型名称		PJScan	ShellOS	MDScan	MPScan	PDFMS	PDF Scrunizer
产生时间		2011	2011	2011	2013	2012	2012
检测方法		静态	动态	静态和动态	静态和动态	静态	静态和动态
检测能力	恶意 Javascript	√	√	√	√	√	√
	代码混淆		√	√	√		√
	反向模拟	√					√
	远程代码加载		√				
	恶意 URI 解析						

由表 3.1 可知，六种检测模型均能够对恶意 Javascript 代码进行有效检测。在六种检测模型中，只有 PJScan 和 PDFMS 使用了静态分析方法，因此，它们无法对混淆后的 PDF 恶意代码进行检测，而动态检测方法不受代码混淆技术的影响，但会产生较高的资源消耗。PJScan 和 PDFScrunizer 对 2013 年提出的新型攻击方法——反向模拟[25]具有较好的检测效果，而其它模型则不能有效检测它。只有 ShellOS 能够对远程代码加载攻击方法予以检测。针对恶意 URI 解析攻击技术，六种模型均无法检测。

经分析，可知现有的 PDF 恶意代码检测模型主要存在以下几点不足：

1）现有模型的检测准确率较低，经测试，PJScan 的检测准确率仅为 72%，ShellOS 的检测准确率约为 80%，不能满足实际需要。

2）现有模型在检测方法上比较单一，都有各自的侧重点，无法应对多种 PDF 攻

击技术。以 PJScan 模型为例，它通过对 PDF 文件中嵌入的 Javascript 代码进行词法分析，完成检测过程。首先，PJScan 无法应对 Javascript 代码混淆技术，其次，该模型不能检测基于非 Javascript 攻击技术的恶意 PDF 文件，因此，漏报率较高。

3）无法在时间开销和检测准确率之间取得平衡。相对于静态检测模型 PJScan，动态检测模型 ShellOS、MDScan 和 PDF Scrutinizer 检测准确率较高，却会产生较大的资源消耗。

4）分类算法选择不当，影响检测结果。PJScan 模型采用的分类算法是 OCSVM（One-Class Support Vector Machine，单类支持向量机）。为构建 OCSVM 算法的分类模型，仅需提供单一类别的 PDF 文件样本集，该算法允许一定比例的样本点位于超球体外部，样本集的数量稀少和种类单一直接影响了 PJScan 的检测准确率。

5）现有模型均采用被动学习策略，它们将人工标注的样本集作为训练样本，以被动的方式接受样本信息，采用基于监督学习的机器学习算法进行分类。现有模型需要较多的训练样本，并需要对各个训练样本进行人工标记，投入的时间和人力成本较大。现有模型不能对最新的恶意 PDF 样本进行有效结合，也不能自适应地更新特征集，因此，模型的特征集很容易过时。

6）检测范围局限。现有模型没有考虑 Adobe Reader 自身的安全漏洞在 PDF 文件攻击中发挥的作用，缺少对 PDF 已知漏洞和 PDF 漏洞利用关键代码的检测。

本文针对现有模型的不足，提出一种新的 PDF 恶意代码检测模型 PDFCheck，该模型结合静态和动态分析方法，主要做了以下改进：将漏洞检测与 PDF 文件文本检测相结合，采用基于主动学习策略的 SVM 分类技术检测 PDF 文本内容，提出一种获取嵌入在 PDF 文件中的 Javascript 代码的方法，对 PDF 已知漏洞类型和漏洞利用关键代码进行检测。

3.2 模型框架设计

本节对上文中提出的 PDF 恶意代码检测模型 PDFCheck 进行框架设计，PDFCheck 模型共包括两个子系统，分别是基于主动学习策略的 PDF 文本检测子系统和 PDF 漏洞检测子系统，如图 3.1 所示。

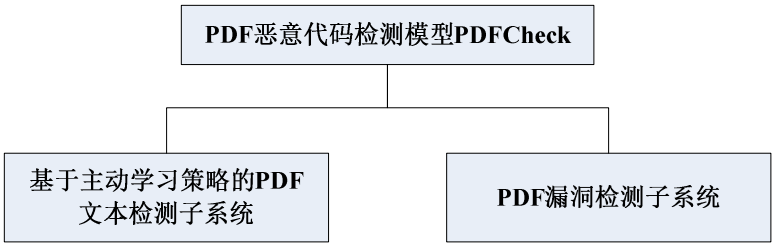

图 3.1 PDFCheck 模型框架

基于主动学习策略的 PDF 文本检测子系统克服了传统检测模型的不足，它通过一些被标签过的新型 PDF 文件（包括正常文件和恶意文件）对检测特征集进行更新。通常，被标签的新型 PDF 文件应能够提升系统的检测能力，其数量是极少的，标签操作一般由安全专家完成。

另外，由于多种常见的 PDF 文件攻击是通过 Adobe Reader 的安全漏洞实施的，PDF 恶意代码也通常是通过漏洞进行传播的，因此，查找 PDF 文件自身存在的安全漏洞，检测 PDF 漏洞利用关键代码（如 ROP 链等），将在根源上对 PDF 恶意代码的传播路径进行阻断，从而更好地应对 PDF 恶意代码的多样性和多变性。如果在 PDF 文件中检测到漏洞利用关键代码，那么此文件将被标识为恶意 PDF 文件。这是我们拟采取的一种新型的 PDF 恶意代码检测思路。

3.3　模型检测流程

3.3.1　基于主动学习策略的PDF文本检测子系统流程

图 3.2 对基于主动学习策略的 PDF 文本检测子系统的流程进行描述，该系统通过检测和标签新型 PDF 文件对检测特征库进行更新，如果被检测的 PDF 文件属性不确定时，应将该文件发送到安全专家处进行分析。

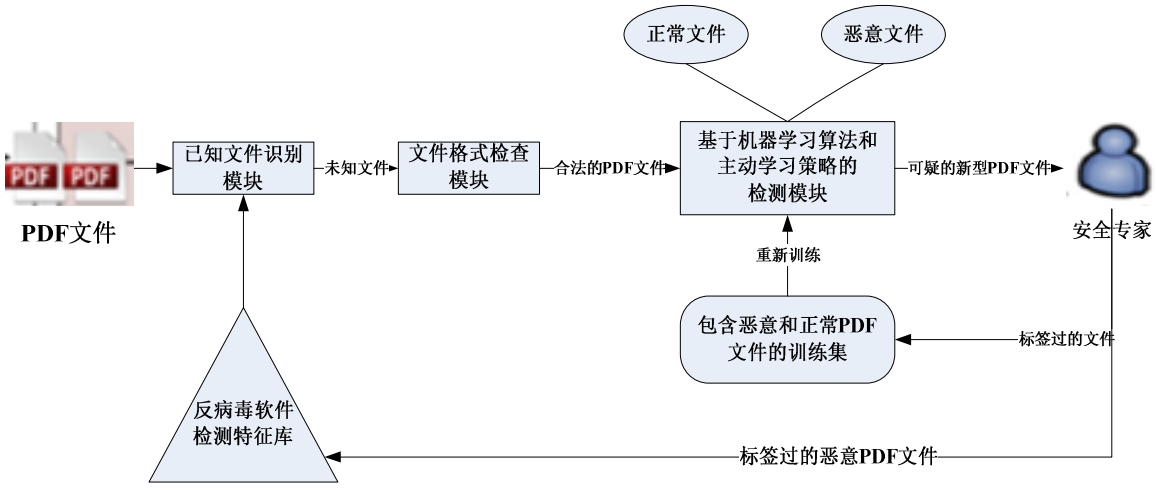

图 3.2　基于主动学习策略的 PDF 文本检测子系统流程

基于主动学习策略的 PDF 文本检测子系统集成了样本集训练和 PDF 文本检测两个主要阶段，其流程描述如下：

1）对经网络传输的 PDF 文件进行收集；

2）已知文件识别模块根据白名单和检测特征库[29]对所有已知的正常和恶意文件进行识别，未知文件则被传送到文件格式检查模块；

3）文件格式检查模块对未知文件进行格式判断，如果该文件不是合法的 PDF 文件，

则会被系统抛弃，只有合法的 PDF 文件才可以进入下一步操作流程；

4）基于机器学习算法和主动学习策略的检测模块对 PDF 文件进行检测，检测时需要通过训练集进行训练，训练集中的数据来自于可信的数据源 Virus-Total 和 Contagio，其中约包含 10000 个恶意的 PDF 文件。一些常见的 PDF 攻击方法，包括本文第二章介绍的 Javascript 代码攻击、嵌入 PDF 攻击以及表单提交和 URI 攻击等都包含在训练集中，内容丰富的训练集对于提升 PDF 恶意代码检测的准确率、减少误报率至关重要。

5）对于可疑的新型 PDF 文件，系统会将其发送到安全专家处进行人工分析；

6）安全专家将标签过的 PDF 文件发送到包含正常文件和恶意文件的训练集中；

7）安全专家将标签过的恶意 PDF 文件发送到反病毒软件检测特征库中。

由上述分析可知，基于机器学习算法和主动学习策略的检测模块是系统的核心模块，该模块主要包括 PDF 文件解析、Javascript 攻击检测、元数据和文件结构特征提取、特征选择以及基于主动学习策略的 SVM 分类等关键步骤，下面对各个步骤进行简述。

1）PDF 文件解析

PDF 文件解析流程包括 PDF 文件类型判断和 PDF 格式解析两个步骤。对 PDF 文件类型的判断主要依据于本文第二章描述的 PDF 文件物理结构，分析输入文件是否具有合法的文件头以及其结构是否完整规范。PDF 格式解析按照页面组、页面、资源、内容等顺序依次解析 PDF 文件，提取 PDF 文件的对象和结构信息，并以树型结构将它们显示出来。

2）Javascript 攻击检测

通过 Hook Adobe Reader 本地 JS 引擎的方式提取 Javascript 源代码，并对其进行堆喷射检测和操作码特征检测，最终输出检测结果。

3）元数据和文件结构特征提取

由 PDF 文件的元数据和文件结构特征构建特征向量集的方法可以减少对特定恶意代码家族和特定攻击方式的依赖。该种方法提取到的元数据和文件结构特征通常是 PDF 文件的文本统计特性，如 font 对象的个数和 stream 对象的平均长度等。

4）特征选择

采用元数据和文件结构特征提取方法会提取到数量繁多的 PDF 文件特征，为取得较好的训练和分类效果，我们需要对特征进行有效选择。测试特征数量改变时检测准确率的变化，从而选定最优的特征数量，并提取最能反映 PDF 文件文本特性的特征。

5）基于主动学习策略的 SVM 分类

将主动学习策略与 SVM 分类技术相结合，对由询问函数选择的 PDF 样本进行人工标记，将其添加到训练样本集中，从而得到更新的训练模型。

3.3.2　PDF漏洞检测子系统流程

PDF 漏洞检测子系统的核心在于漏洞原理分析、已知漏洞检测和漏洞利用关键代码检测，其流程如图 3.3 所示。

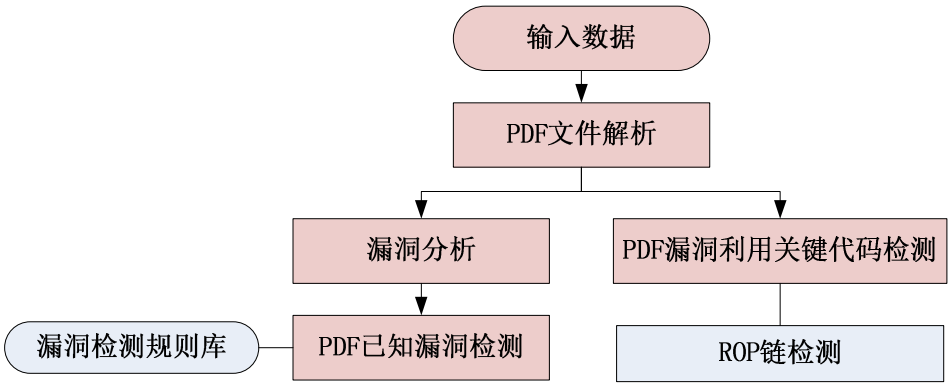

图 3.3　PDF 漏洞检测子系统流程

PDF 已知漏洞检测和 PDF 漏洞利用关键代码检测都是以 PDF 文件解析为基础的，只有将 PDF 文件的结构、对象、字段等按序提取出来，才能进行后续操作。通过对大量 PDF 漏洞样本进行调试分析，研究漏洞的触发原理、产生条件和检测方式，搜集、抽取、存储和整理漏洞特征，归纳得到已知 PDF 漏洞检测规则特征库，将特征库中的检测规则与待查 PDF 样本进行特征匹配，如果 PDF 样本中有一个或多个数据结构的值命中了特征库中的一条或多条检测规则，则判定该样本文件存在 PDF 已知漏洞。PDF 漏洞利用关键代码包括 ROP 链等，ROP 链主要用于突破 DEP（Data Execution Prevention，数据执行保护）防护机制，它本身是一些指令片段（gadget），这些片段的结尾都是 ret 指令。

3.4　模型评价指标

现有的恶意代码检测模型都有其特定的评价指标。我们首先对几个常见的模型评价术语进行介绍，假设样本集总数是 M，则检测结果的分类如表 3.2 所示[30]。

表 3.2　检测结果分类表

	实际正常样本数量	实际恶意样本数量
检测正常样本数量	*TP*	*FP*
检测恶意样本数量	*FN*	*TN*

其中，$M=TP+FP+FN+TN=P+N$。$N=FP+TN$，代表实际恶意样本的数量。$P=TP+FN$，代表实际正常样本的数量。

PDF 恶意代码检测模型 PDFChecker 的评价指标主要有七种：

1）*Accuracy*（准确率）：是最常用的评价指标，表示检测正确的样本数与所有样本数的比值，*Accuracy*=(*TP*+*TN*)/(*P*+*N*)，通常该值越大，检测效果就越好。

2）*Error Rate*（错误率）：与 *Accuracy* 相反，表示检测错误的样本数与所有样本数的比值，*Error Rate*=(*FP*+*FN*)/(*P*+*N*)，且 *Accuracy=1-Error Rate*。

3）*Specificity*（特定度）：表示恶意样本中被正确检测的样本数与恶意样本总数的比值，*Specificity=TN/N*，该值可以衡量模型对恶意代码的识别能力。

4）*Sensitive*（灵敏度）：表示正常样本中被正确检测的样本数与正常样本总数的比值，*Sensitive= TP/P*，该值可以衡量模型对正常样本的识别能力。

5）*Precision*（精度）：精度是对模型检测精确性的度量，表示被检测为正常样本的样本数中实际正常样本所占的比例，*precision=TP/*(*TP*+*FP*)。

6）*TPR*（True Positive Rate）：表示预测结果是正常样本，而且也确实是正常样本的概率，*TPR=TP/*(*TP*+*FN*)。

7）*FPR*（False Positive Rate）：表示预测结果是正常样本，但实际为恶意样本的概率，*FPR=FP/*（*TN*+*FP*）。

3.5 模型测试数据与实验环境

3.5.1 数据收集

本文的训练数据集来自于可信数据源VirusTotal repository[31]、Contagio Project以及Internet and Ben-Gurion University，表3.3对三个可信数据源进行介绍。通过上述数据源，我们共收集到恶意PDF样本5157个，收集到正常PDF样本3989个。其中，3989个正常PDF样本已经通过卡巴斯基实验室的反病毒检测；恶意PDF样本则涉及Javascript代码攻击、嵌入文件攻击以及表单提交和URI攻击等多项攻击技术，其中有一些样本被混淆。

表 3.3 可信数据源简介

数据源	时间	恶意样本数量	正常样本数量
VirusTotal repository	2012-2014	17596	0
Contagio Project	2012	410	0
Internet and Ben-Gurion University	2013-2014	0	5145

在已收集的9146个PDF样本中，选取训练样本1500个，剩余的则为预测样本。训练样本中的恶意PDF文件有1000个，预测样本中的恶意PDF文件有4157个。训练样本和预测样本的组成结构如表3.4所示。

表 3.4　训练样本和预测样本的组成结构

	训练样本	预测样本
正常样本	500	3489
恶意样本	1000	4157
总数	1500	7646

3.5.2　实验环境

本文在个人 PC 上对基于主动学习策略的 PDF 文本检测子系统和 PDF 漏洞检测子系统进行功能和性能测试，其环境参数如表 3.5 所示。

表 3.5　环境参数

测试实验环境	内容
机器数量	联想 Idealpad Y470，笔记本电脑 1 台
机器配置	CPU：Intel Core i3，内存：4G DDR3
操作系统	Windows 7 旗舰版（64 位）

3.6　本章小结

本章首先分析现有检测模型的缺点，并针对其不足提出一种新的 PDF 恶意代码检测模型 PDFCheck，对模型的整体框架和主要流程进行说明，然后介绍 PDF 恶意代码检测模型的评价指标、测试数据和实验环境。

第四章　基于主动学习策略的 PDF 文本检测子系统

4.1　系统总体架构

基于主动学习策略的 PDF 文本检测子系统通过将新型的恶意 PDF 文件添加到检测特征集中，使系统具有自适应和主动学习的能力，其主要流程如图 3.2 所示。系统主要包括 PDF 文件解析、Javascript 攻击检测、元数据和文件结构特征提取、特征选择以及基于主动学习策略的 SVM 分类等关键模块，其总体架构如图 4.1 所示。

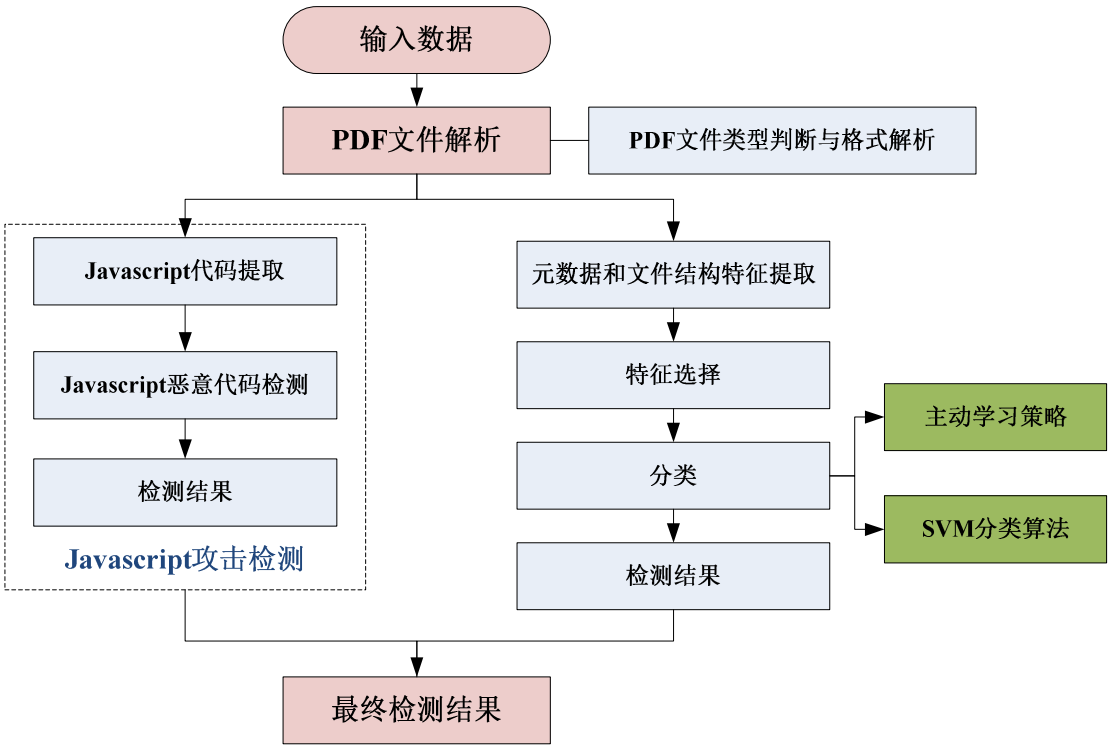

图 4.1　基于主动学习策略的 PDF 文本检测子系统总体架构

4.2　PDF 文件解析

PDF 文件解析模块包括 PDF 文件类型判断和 PDF 格式解析两个步骤。

1）PDF 文件类型判断

对 PDF 文件类型的判断主要依据本文第二章描述的 PDF 文件物理结构，分析输入文件是否具有合法的文件头以及其结构是否完整规范。应首先采用字节流的形式将 PDF 样本中所有数据结构的值映射到内存，通过特征匹配的方法判断该样本是否为合法的 PDF 文件，判断依据主要是 PDF 文件中的关键字段，如文件头的%PDF-[version number]、文件尾部“startxref”和“%%EOF”之间的数字等。在 PDF 文件中，“startxref”

和“%%EOF”之间包含了最后一个交叉引用表的偏移量，如果该值不正确，那么 PDF 文件也是不合法的，应将其过滤掉。

2）PDF 格式解析

解析 PDF 文件时应首先通过 startxref 对根对象所在的 xref 的位置进行查找，然后由 trailer 得到根对象的序号，此时就精确定位到了 Catalog 根节点。接下来根据交叉引用表对 PDF 文件中的间接对象进行访问，按照页面组、页面、资源、内容等顺序依次解析 PDF 文件，提取 PDF 文件的对象和结构信息。

PDF 格式解析的具体流程[32]如图 4.2 所示。

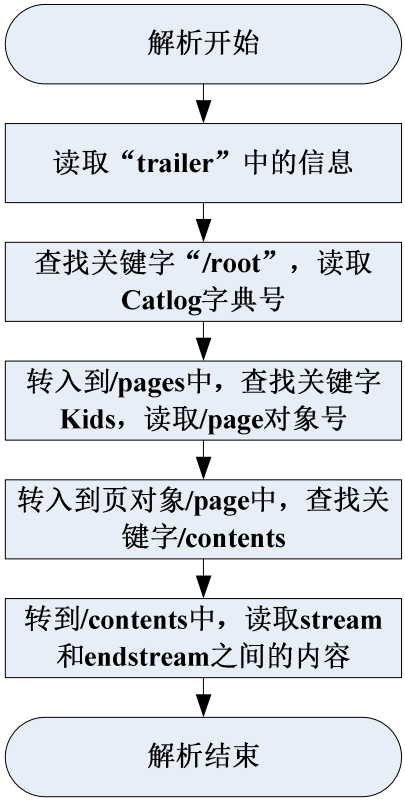

图 4.2　PDF 格式解析流程

4.3　Javascript 攻击检测

Javascript 攻击检测模块主要包括 Javascript 代码提取和 Javascript 恶意代码检测两个步骤，其总体框架如图 4.3 所示。通过 Javascript 代码提取步骤，可以获得 Javascript 源代码和操作码，输出的源代码和操作码将被用于恶意代码检测步骤的输入数据。

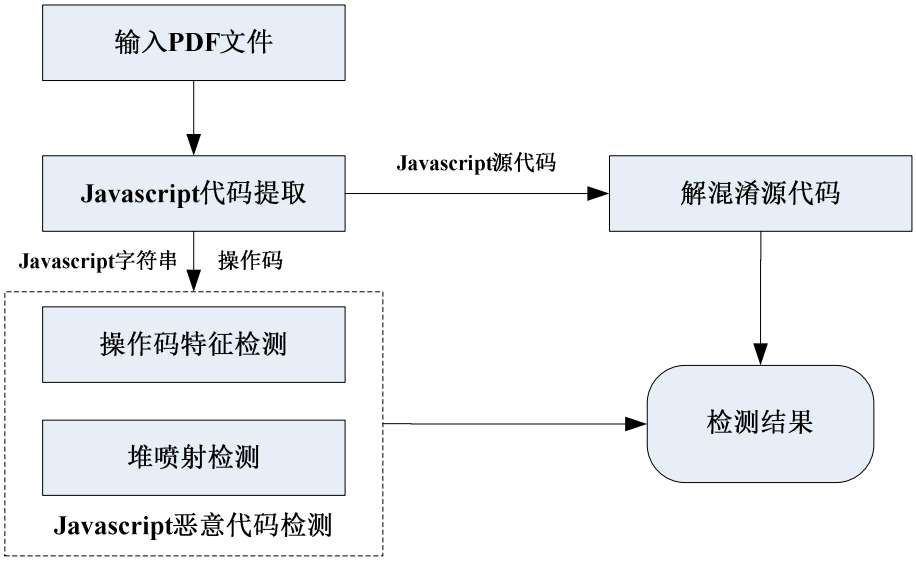

图 4.3　Javascript 攻击检测总体框架

4.3.1　Javascript代码提取

1）静态 Javascript 代码提取方法

Javascript 代码提取的前提是代码定位，首先定位 PDF 文件中的 Javascript 代码，分析 Javascript 代码存在于哪个对象中，按照对象之间的引用关系，从对象中提取 Javascript 代码，并将其存储到新的文本文件中。

由于 Javascript 代码可以出现在 PDF 文件的多个对象中，并且攻击者能够采用间接引用的方式，因此，在 PDF 文件中提取 Javascript 代码是十分困难的。在 PDF 文件中，Javascript 代码通常存在于 dictionary，dictionary 是以“<< ”和“>>”包含的若干条目，各个条目都是由 key 与 value 两部分构成，其中，key 具有唯一性。

Javascript 代码通常嵌入在关键字是“/JS”的 dictionary 中，JavaScript 代码有两种嵌入方式：一种是文本方式或十六进制的字符串，另一种是指向包含 Javascript 代码的字符串的间接引用。如果采用第二种嵌入方式，可以将含有关键字“/JS”的 dictionary 对象放置到 stream 对象中，并使其在 stream 对象中被压缩或加密，关键字“JS”被 stream 压缩后是不可见的纯文本。PDF 文件中 Javascript 代码的两种嵌入方式如图 4.4 所示。

```
1 0 obj                    1 0 obj
<< /Type /Catlog           << /Type /Catlog
   /OpenAction <<             /OpenAction <<
    /S /Rendition              /S /Javascript
    /JS 16 0 R                 /JS (alert( 'Hello!' );)
    >>                         >>
 >>                         >>
```

图 4.4　Javascript 代码两种嵌入方式

通常，PDF 文件中的 JavaScript 代码必须被包含于 action dictionary，action dictionary 有两个重要标识：/JavaScript 和/Rendition，它们都含有关键字“/JS”，因此，可以根据/JavaScript 和/Rendition 对 JavaScript 代码的入口位置进行定位。/Javascript 和/Renditon 可能在以下位置被找到[20]：

①Catalog dictionary 的/AA 入口；

②Catalog dictionary 的/OpenAction 入口；

③文件的 name 入口；

④文件的 outline 层次可能包含对 Javascirpt action dictionary 的引用；

⑤页面、文件附件和表单可能包含对 Javascirpt action dictionary 的引用。

2）现有 Javascript 代码提取方法的不足

JavaScript 代码除了嵌入到 PDF 文件以外，还可能存在于本地主机的其他文件中，或者驻留在远端主机上，可以通过/URI 或/GoTo 指令跳转到远端主机。使用函数 setTimeOut()或 eval()也可以对 Javascript 代码进行动态调用。另外，针对 Javascript 代码可以采用混淆技术。当前最常见的 Javascript 代码混淆技术有字符串分隔技术和添加随机注释技术两种，对一段 Javascript 代码添加随机注释的示例如图 4.5 所示。在图 4.5 中添加了大量的无关注释，例如/* jfjsfgs sfg sfsgg sllgarg ffg ggh hhhhhhh afdf */，这些注释代码会增加 Javascript 代码的提取难度。

```
Cvb345 = "Num"+"ber"+"s:"; For (i=0;i<10;i++)
/* jfjsfgs sfg sfsgg  sllgarg ffg ggh hhhhhhh
afdf */ { /* ksgdgh lldlf */ Cvb345 = Cvb345 +
i.toString(); /* swrfdg sgg gsgsllllghhs ssffg
jsgset */ }
/* new line comment */
```

图 4.5　Javascript 代码添加随机注释示例

PDF 恶意代码检测模型 MDScan 采用静态分析方法，通过构建一个 PDF 文件解析器对嵌入的 Javascript 代码进行寻找，由于需要模仿大量的 Acrobat Javascript API，这种方法很难覆盖到所有针对 PDF 文件的混淆技术。即使通过这种方法得到了 Javascript 代码片段，如何将它们按照正确的序列进行放置，对静态分析方法也是有挑战的。

PJScan[9]通过 Hook SpiderMonkey JS 引擎的方式获得令牌流，由于 SpiderMonkey[33]的 JS 引擎是一个模拟环境，它缺少一些本地 Adobe 环境中的特征，会导致一些非预期的输出结果。因此，PJScan 无法提取 PDF 文件中的所有 Javascript 代码。

3）本文提出的 Javascript 代码提取方法

在介绍本文提出的 Javascript 代码提取方法之前，首先对 Adobe Reader 本地 JS 引擎的架构进行描述。Adobe Reader 的本地 JS 引擎是一个基于操作码的解释器，它具有词法分析、语法分析、生成操作码、优化中间代码和生成目标代码等功能，其架构[34]

如图 4.6 所示。

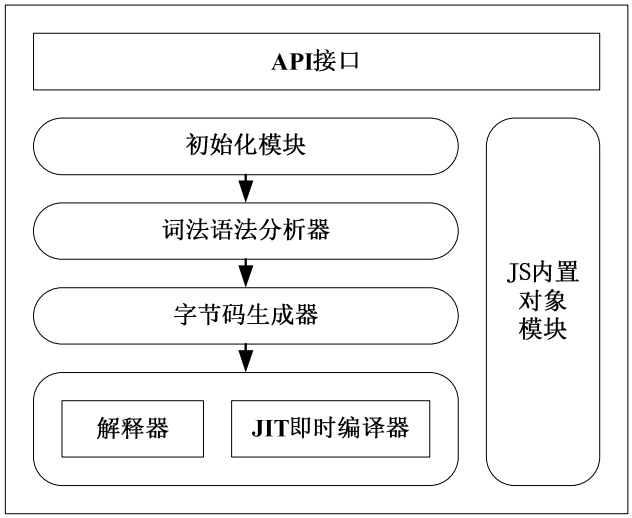

图 4.6　Adobe Reader 本地 JS 引擎架构

本文提出的 Javascript 代码提取方法通过 Hook Adobe Reader 的本地 JS 引擎，对 Javascript 代码进行提取，从而有效抵御未知的 Javascript 代码混淆技术，处理流程如图 4.7 所示。

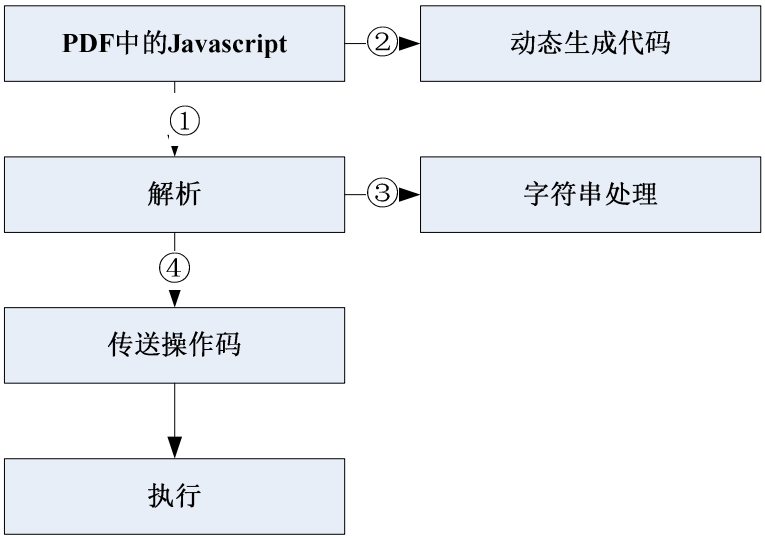

图 4.7　Javascript 代码提取流程

在图 4.7 中，步骤①是解析的起始点，所有 Javascript 源代码在执行之前必须被解析；步骤②用于处理由方法 app.eval()和 new function()动态生成的 Javascript 源代码，将步骤①和步骤②的结果结合起来，对其进行 hook 操作，可以得到完整的解混淆后的 Javascript 代码；步骤③表示 Javascript 字符串被生成和处理，通过 hook 对应函数，可以直接提取 Javascript 字符串；步骤④表示操作码执行，每一个操作码在一个类似于 switch()的结构中被处理，通过 hook 它，我们可以提取操作码流。

由于 Adobe Reader 是闭源软件，我们需要借助逆向工程技术去定位这些 hook 点。图 4.7 中步骤①的 hook 点是 Adobe Reader JS 引擎中的 parser()函数，该函数用于对源程序进行词法分析和语法分析；步骤②的 hook 点是 JS_EvaluateScript()函数，步骤③的 hook 点是 JS_NewStringCopyN()和 JS_NewStringCopyZ()函数；步骤④的 hook 点是 js_EmitTree()函数，该函数用于生成操作码。

当嵌入在 PDF 文件中的 Javascript 代码执行时，可以提取到 Javascript 源代码、字符串以及操作码，最终得到的 Javascript 源代码和操作码具有正确的执行序列。

4.3.2 Javascript恶意代码检测

Javascript 恶意代码检测模块主要包括堆喷射检测和操作码特征检测两个组件。

1）堆喷射检测

堆喷射技术被恶意 PDF 文件用于操作内存堆，它通常与堆溢出成对出现，其详细介绍见本文 2.2.1 节。String 数据类型经常被用于装载堆喷射代码和 Shellcode，这是因为在 Javascript 中，String 是仅有的不被垃圾回收的数据类型。

为对堆喷射进行检测，首先通过图 4.7 中步骤③的操作将 Javascript 字符串提取出来，并对其中长度大于 64K 字节的字符串进行选择。这是因为 64K 字节以下的字符串长度较小，不适用于堆喷射技术。

下面通过计算字符串熵值的方式检测堆喷射。由于堆喷射代码中包含多个重复字符，其熵值远远低于正常的字符串。Zhijie Cetal 经研究表明将熵阈值设置为 1 将获得最好的检测效果。因此，在堆喷射检测组件中，将熵值设置为 1，这就意味着任何熵值小于 1 的字符串将被标记为堆喷射代码。

计算字符串信息熵的公式描述如下：

$$H(x)=E\left[\log\frac{1}{p(ai)}\right]=-\sum_{i=1}^{n}p(ai)\log p(ai) \qquad (4\text{-}1)$$

其中 $p(ai),i=1,2,...,n$ 表示字符串取第 i 个符号的概率，$H(x)$ 反映了整个字符串的统计特性。对于特定的字符串，其信息熵的值是唯一的。

接下来我们使用本文提出的方法对 PDF 漏洞 CVE-2010-3654 进行检测，CVE-2010-3654[35]是一个存在于 authplay.dll（AuthPlayLib.bundle 或 libauthplay.so.0.0.0）中的缓冲区溢出漏洞。通过该漏洞，远程攻击者可利用特制的 SWF 内容去执行任意代码或导致拒绝服务。CVE-2010-3654 利用嵌入在 PDF 中的 Flash 文件完成对堆的操纵，漏洞利用关键代码如下所示：

```
<script>
function HeapSpray(){
  shellcode=unescape("%u68fc%u0a6a%u1e38%u6368%ud189%u684f%u7432%u0c91%uf48b%u7e8d
```

```
  %u33f4%ub7db%u2b04%u66e3%u33bb%u5332%u7568%u6573%u5472%ud233%u8b64%u305a%u4
  b8b%u8b0c%u1c49%u098b%u698b%uad08%u6a3d%u380a%u751e%u9505%u57ff%u95f8%u8b60%
  u3c45%u4c8b%u7805%ucd03%u598b%u0320%u33dd%u47ff%u348b%u03bb%u99f5%ube0f%u3a06
  %u74c4%uc108%u07ca%ud003%ueb46%u3bf1%u2454%u751c%u8be4%u2459%udd03u8b66%u7b3
  c%u598b%u031c%u03dd%ubb2c%u5f95%u57ab%u3d61%u0a6a%u1e38%ua975%udb33%u6853%u
  6577%u7473%u6668%u6961%u8b6c%u53c4%u5050%uff53%ufc57%uff53%uf857");
  block = unescape("%u1212%u1212");
  while (block.length < 0x100000)
    block += block;
  block = block.substring( 0, 0x100000/2-32/2-4/2-2/2-shellcode.length-16 );
  block += shellcode;
  var memory = new Array();
  for (i = 0 ; i < 600 ; i++)
    memory[i] = block.substring(0, block.length);
}
……
</script>
```

使用堆喷射检测组件对包含上述代码的 PDF 样本进行检测，可知 memory 字符串的长度约为 200M，计算该字符串的熵值，远远小于 1，因此，该字符串被标记为堆喷射代码。

2）操作码特征检测

操作码是由 Javascript 引擎生成的中间指令，它比源代码低一个层级，会真正影响恶意代码的行为。无论源代码级别的恶意 Javascript 代码是如何构建的，它们都会有一些具体的行为特征，包括漏洞利用、从远端主机获得文件等。因此，对恶意 Javascript 操作码的流模式进行特征匹配是识别 Javascript 恶意代码的有效方法。

当不同的 Javascript 代码触发同一个漏洞时，操作码特征检测是很有用的。图 4.8 中的两个 Javascript 代码片段在文本级具有不同的外在形式，却都可以通过 getIcon()方法的栈溢出触发 CVE-2009-0927 漏洞，它们的操作码是相同的，如图 4.9 所示。

```
代码段 1:
Collab.getIcon(var_x);

代码段 2:
var geticon = new Function("arg", "return Collab.getIcon(arg);");
geticon(var_x);
```

图 4.8　触发同一个漏洞的两段 Javascript 代码

```
Opcode: 184 - getmethod
Opcode: 154 - getgvar
Opcode: 58 - call
```

图 4.9　两段 Javascript 代码生成的操作码

此时我们可以构建一个基于操作码的确定性有限自动机，通过图 4.10 中的自动机状态转换，对触发 CVE-2009-0927 漏洞的 Javascript 恶意操作码进行特征匹配。

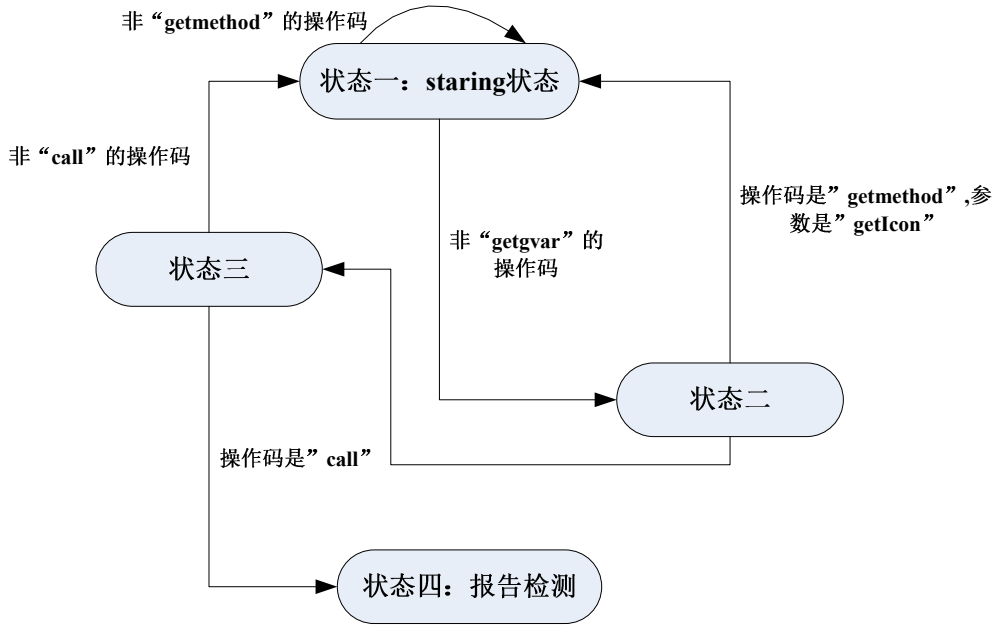

图 4.10　CVE-2009-0927 漏洞利用特征匹配

4.4　元数据和文件结构特征提取

本文采用基于元数据和文件结构的特征提取[11]方法对 PDF 文件进行特征提取，该方法与特定的漏洞和具体的攻击方式是无关的。它通常采用简单字符串匹配的方式，提取 PDF 文件的文本统计特性，例如统计 font 对象的个数和 stream 对象的平均长度等，提取到的特征均为数字。为统计 stream 对象的长度，只需要在文本中定位“stream”标志和接下来最近的“endstream”标志，两个标志之间的部分即为 stream 对象。为进一步识别和提取原始 PDF 文件中的数据，也可以使用正则表达式。

这种快速和松散的元数据提取方法可能会导致一些不准确或模糊的结果数据出现，但最终的特征是确定的。例如，对一个不断被编辑和更新的 PDF 文件进行特征提取，提取到的对象数量会一直增多。另外，当元数据的多个实例项在 PDF 文件中重复出现时，不使用正确的值，而使用一些其它的特征，例如不同的值被使用的次数。

对于加密的 PDF 文件，仍然可以使用该项特征提取技术，虽然各个对象或 stream 分别被加密了，但是它们的结构没有发生变化，依旧可以提取到 PDF 文件的元数据和结构特征。

4.5　特征选择

通过基于元数据和文件结构的特征提取方法，我们可以提取到大量的 PDF 文件特征，本节讨论如何对这些特征进行选择才能取得较好的分类效果。

对于分类算法来说，特征提取至关重要，具有区分性的特征能够显著提升分类效果，而标定错误或包含噪声的样本却会带来分类效率的下降。

特征选择的目的在于尽可能地用参数表示 PDF 文件的元数据和结构，但需要去除无用的特征。能够反映元数据特性的特征有：每个域中字符个数，对象、stream 的大小与个数，图像的大小与位置，数据加密算法的使用次数以及加密对象的个数等。经选择，共有 202 个特征，按重要程度对这些特征进行排序，其中排名前列的依次是：count_font、count_javascript、count_js、count_stream_diff、pos_box_max、image_totalpx、producer_len、count_obj、pdfid0_mismatch、pos_eof_max、len_stream_min 等。图 4.11 对各个特征的重要性进行说明。其中，count_font 特征是通过关键字“/Font”获取的，count_javascript 特征是通过关键字“/Javascript”和“/JS”获得的，count_obj 表示 obj 关键字的个数。

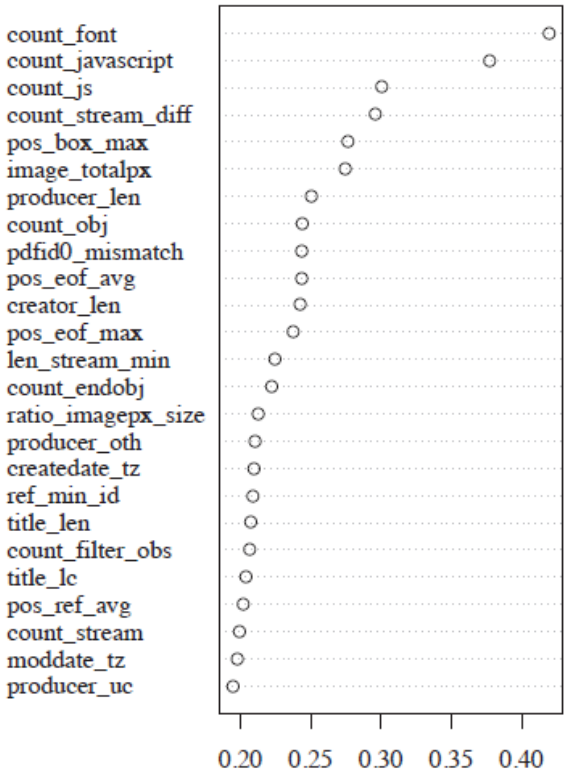

图 4.11　元数据和文件结构特征的重要性比较

下面对选取的 202 个元数据和文件结构特征进行性能评估，测试特征的合理性和有效性。

由恶意训练集决定的对象个数是 10354，恶意训练集中单个对象的最高出现频率是

5945。图 4.12 描述了恶意训练集聚类的结果，图中 y 轴表示对象的频率。在图 4.12 中，有 28 个对象出现次数最多，具有较高的频率，它们被聚类器认为是特殊的。

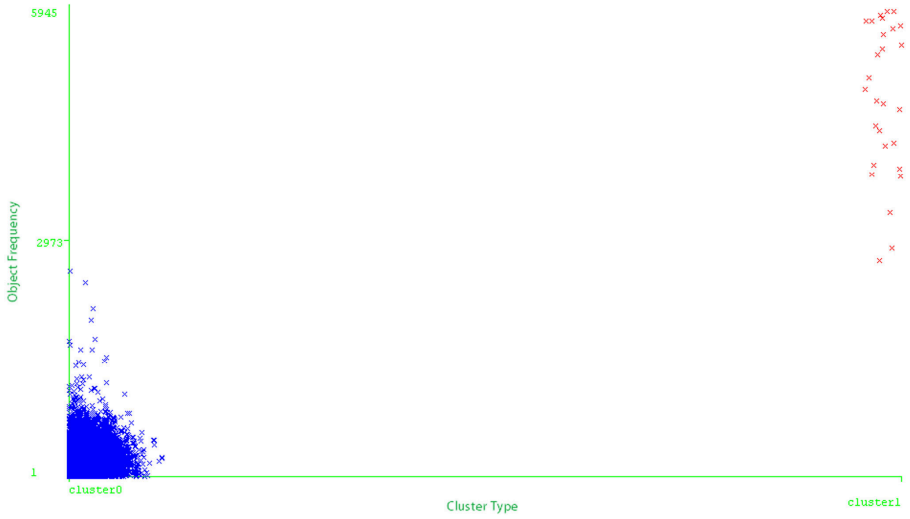

图 4.12　恶意训练集聚类

由正常训练集决定的对象个数是 650357，单个对象的最高出现次数是 5965。相比于恶意训练集，正常训练集中包含的对象种类更多。图 4.13 描述了正常训练集的聚类结果。在图 4.13 中，有 156 个对象具有较高的频率，它们被聚类器认为是特殊的。最终，将从与正常文件相关的簇中获得的对象与从恶意文件相关的簇中获得的对象进行合并，这个操作获取到了 168 个对象。

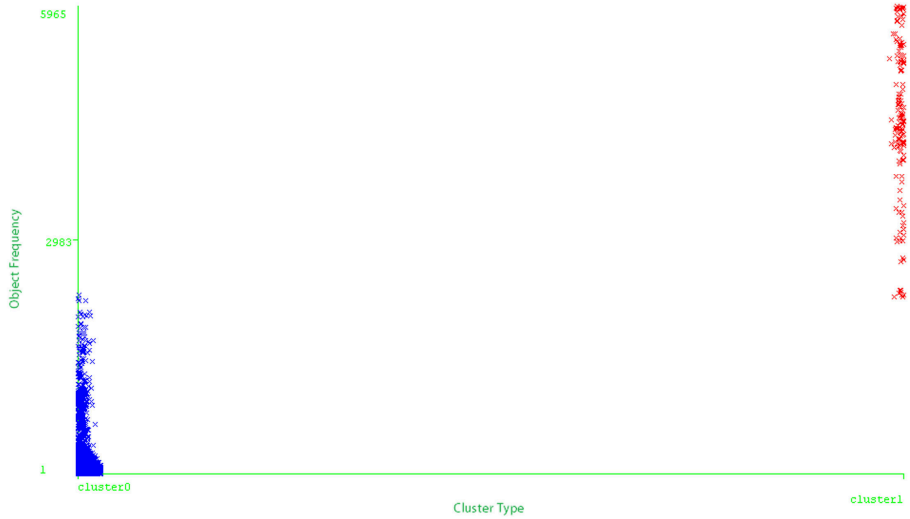

图 4.13　正常训练集聚类

由图 4.12 和图 4.13 可知，恶意文件训练集和正常文件训练集的聚类结果存在很大差异，这表明本文选定的特征向量是较为合理的。

基于元数据和文件结构的特征提取方法可以获得大量的 PDF 文件特征，而特征数量会影响分类的准确率。通过测试特征数量改变时分类错误率的变化情况，可以得到表 4.1 中的统计数据，其中分类采用 SVM 算法。

表 4.1　特征数量对分类错误率的影响

特征数量	分类错误率（Error Rate）
10	8.4%
20	7.8%
30	4.97%
40	4.48%
50	3.86%
60	3.29%
70	2.13%
80	1.91%
90	1.56%
100	1.73%
110	1.62%
120	1.09%
130	1.12%
140	1.18%
150	0.96%
160	1.14%
170	0.89%
180	0.94%
190	0.99 %
200	1.02 %

由表 4-1 可知，特征数量较少时，分类错误率相对较高，当特征数量达到 120 时，随着特征数量的增多，分类错误率始终较低且变化不大。因此，应该依据重要性选取 PDF 文件的前 120 个特征作为特征向量。

4.6 分类

4.6.1 SVM分类算法

1）选择 SVM 分类算法的原因

SVM（Support Vector Machine，支持向量机）是一种具有良好效果的分类预测算法，它采用 RBF（Radial Basis Function，径向基核函数）完成分类过程。选择 SVM 作为分类算法主要有以下几方面的原因：第一，Srndic and Laskov[10]、Maiorca[12]、Borg[36]和 Schreck[37]等人的研究工作表明，对于恶意 PDF 文件检测，尤其是当提取的文件特征较多时，SVM 是一个非常精确的分类器，相比于其它分类算法，SVM 效果更为突出，其 *TPR* 可以高达 0.998；第二，该算法基于 RBF 内核训练 SVM 分类器，将数据集投射到高维空间中，并对分类超平面的线性分隔进行计算，从而分离两类数据。这种向高维空间投射数据集的方式使模型结构复杂，因此，攻击者很难理解该模型，另外，通过寻找特定特征或模式来规避 SVM 分类模型也是困难的[38]；第三，SVM 具有处理大量文本特征的能力，它被成功地用于蠕虫检测，SVM 算法具有效率高、性能优越等特点，适用于 0-day 攻击检测[39]；第四，SVM 算法可以与主动学习策略相结合，这会使其检测恶意代码的能力更强。对 SVM 分类算法的实现可以通过 Libsvm 库[40]，也可以使用 C#编程语言的 SVM 类库。

2）SVM 分类算法原理

现在我们简单介绍 SVM 分类算法的原理：

当前的 SVM 分类器主要有两种：a.一对一：即二类 SVM 分类器；b. 一对多：构造多个二类 SVM 分类器，使每个类对应于其中一个。SVM 采用结构风险最小化原则来对分类问题进行处理。

设样本集 S 线性可分，$S=\{(x_i, y_i) \mid i=1,...,n\}$，其中 $x_i \in R^d$，$y_i \in \{1,-1\}$，y_i 代表 x_i 对应的类别，如果 x_i 属于第一类，则将它标记为正样本（y_i=1），如果 x_i 属于第二类，则将它标记为负样本（y_i=-1）。

如果样本集是线性可分的，那么在两类样本中间有一个隔离带，以二维空间的样本对其进行表示，如图 4.14 所示。

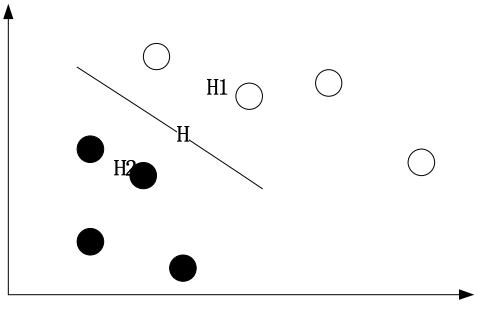

图 4.14 二维空间样本线性可分

在图 4.14 中，黑点和白点分别表示两类不同的训练样本，H 将它们分开，而 H_1 和 H_2 与 H 是平行的，并且与 H 之间具有相等的距离。H_1 和 H_2 之间的区域是分开两类样本的隔离带，设 H_1 和 H_2 之间的距离是 2△，那么△为分类间隔。第一类样本中距 H 最近的点停留在了 H_1 上，第二类样本中与 H 距离最近的点停留在了 H2 上，H1 和 H2 上的点都位于隔离带的边界上，它们被称作支持向量。构造决策函数的目的是正确地分隔测试样本。d 维空间中的线性判别函数通常表示为 $g(x)=wx+b$，因此，分界面 H 可表示为：

$$wx+b=0 \tag{4-2}$$

如果分界面 H 能对两类样本进行正确分类，使

$$\begin{cases} wx+b\geq 1 \text{若} \mathrm{y}_i=1 \\ wx+b\leq 1 \text{若} \mathrm{y}_i=-1 \text{其中} \mathrm{i}=1,2,\ldots,\mathrm{n} \end{cases} \tag{4-3}$$

那么就称样本集是线性可分的。

H_1 和 H_2 之间的距离是 $2\|w\|$，它们之间的距离是与 $\|w\|$ 成反比的。通过公式(4-2)对 $g(x)$ 做归一化操作，使两类样本都满足下述条件：$|g(x)|\geq 1$，这时候的分类间隔是 $2/\|w\|$。$\|w\|$ 越大，则分类间隔的值越小。为使分类面能够对全部样本进行正确分类，需满足下述条件：

$$y_i[(wx)+b]-1\geq 0, i=1,2,...,n \tag{4-4}$$

使得上述条件均获得满足的分类面被称为最优分类超平面，如图 4.15 所示。最优分类面问题等价于在满足式（4-4）的条件下，对目标函数：

$$\phi(w)=\frac{1}{2}\|w\|^2=\frac{1}{2}(w\cdot w) \tag{4-5}$$

的最小值进行求解。

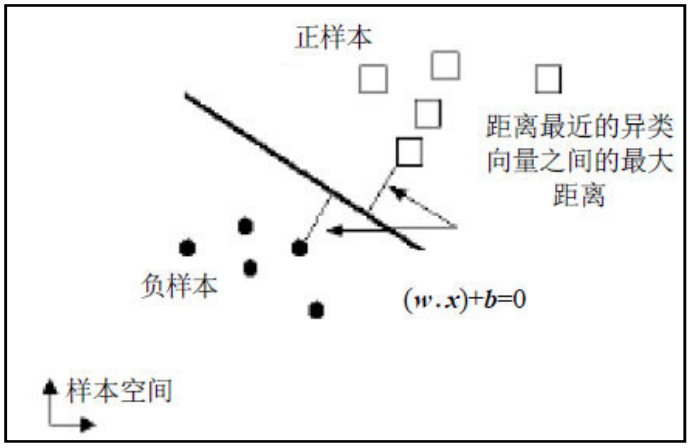

图 4.15　最优分类超平面

此时，对线性支持向量机的求解可以转化为线性规划问题：

$$\begin{cases} \text{目标函数:}\min\dfrac{\|w\|^2}{2} \\ \text{约束条件：} y_i[(wx)+b]-1\geq 0, i=1,2,...,n \end{cases} \tag{4-6}$$

对于线性不可分样本，引入松弛变量ξ_i和惩罚因子C，此时式（4-6）可改写为：

$$\phi(w,\xi_i)=\frac{1}{2}(w\cdot w)+C(\sum_{i=1}^{N}\xi_i) \tag{4-7}$$

此时，对拉格朗日乘子$\alpha_i(i=1,2,...,n)$进行引入，从而将 SVM 分类转化为有约束的二次函数极值问题，然后对最优的分类面进行求解。最终解为

$$w=\sum_i \alpha_i y_i x_i \tag{4-8}$$

通过公式（4-8）可以求出权向量w的值，然后通过公式（4-2）求出b的值，则决策函数可进一步重写为：

$$f(x)=sign\{(w\cdot x)+b\}=sign\{\sum_{i=1}^{N}\alpha_i y_i(x_i\cdot x)+b\} \tag{4-9}$$

3）核函数

对于函数$K(x,y)$来说，若存在非零函数$\varphi(x)$和实数λ使得

$$\int_a^b K(x,y)\varphi(x,y)dx=\lambda\varphi(y) \tag{4-10}$$

成立，那么就将 $K(x,y)$ 称作核函数，将 λ 称作核函数的特征值，将 $\varphi(x)$ 称为核函数关于 λ 的特征函数。

常见的核函数包括以下四种：

① Linear 核

$$K(\vec{x},\vec{y})=\left\langle \vec{x},\vec{y} \right\rangle \tag{4-11}$$

② Puly 核

$$K(\vec{x},\vec{y})=(\gamma\left\langle \vec{x},\vec{y} \right\rangle+r)^4 \tag{4-12}$$

③ RBF 核

$$K(\vec{x},\vec{y})=\exp(-\gamma\|\vec{x}-\vec{y}\|^2) \tag{4-13}$$

④ Sigrnoid 核

$$K(\vec{x},\vec{y})=\tanh(\gamma\left\langle \vec{x},\vec{y} \right\rangle+r) \tag{4-14}$$

对核函数的选用将会影响 SVM 分类器的效果[41]。通常，RBF 核是我们的优先选择，它以非线性的形式将样本映射到多维空间中，能够对属性和分类标注的非线性关系进行处理。但是当特征维数比较大时，只能使用 Linear 核。本文提取的 PDF 元数据和文件结构特征仅有 120 个，适用于 RBF 核。

4）参数选择

SVM 分类算法的一个重要参数是惩罚因子 C，C 的值可以选择 10^t，其中 $t\in\{-4,-3,-2,-1,0,1,2,3,4\}$，也就是 0.0001 到 10000 之间。$C$ 的值越大，表示对错误的惩罚程度越严重，C 值过大有可能造成模型过拟合。下面通过交叉验证法对参数 C 的值进行计算：

①对已标记的训练样本进行等分，将其分为 N 个训练子集，其中每个训练子集中至少含有一个负样本和一个正样本；

②选取 N-1 个子集进行训练，将剩下的一个子集加入测试集，通过 SVM 分类器对测试集进行预测；

③重复进行步骤②中的操作，直到全部样本均被预测一次，对每个子集上测得的类别信息进行综合，进而获得整个训练集的类别信息；

④将原类别信息与 SVM 分类器测得的类别信息进行对比，得出最终的准确率

Accuracy。

选取不同的惩罚因子 *C* 进行测试，输出结果如图 4.16 所示。

```
C:\Users\Administrator\Desktop>python test.py
Set C=0.0001 Accuracy=0.854
Set C=0.001 Accuracy=0.962
Set C=0.01 Accuracy=0.977
Set C=0.1 Accuracy=0.973
Set C=1 Accuracy=0.979
Set C=10 Accuracy=0.981
Set C=100 Accuracy=0.996
Set C=1000 Accuracy=0.977
Set C=10000 Accuracy=0.935
```

图 4.16　C 参数值测试结果

对采用不同参数 *C* 得出的准确率 *Accuracy* 进行比较，如图 4.17 所示。图中，*C* 参数值为 100 时，准确率最高。因此，*C*=100 是最佳参数，此时，SVM 分类器的性能是最优的。

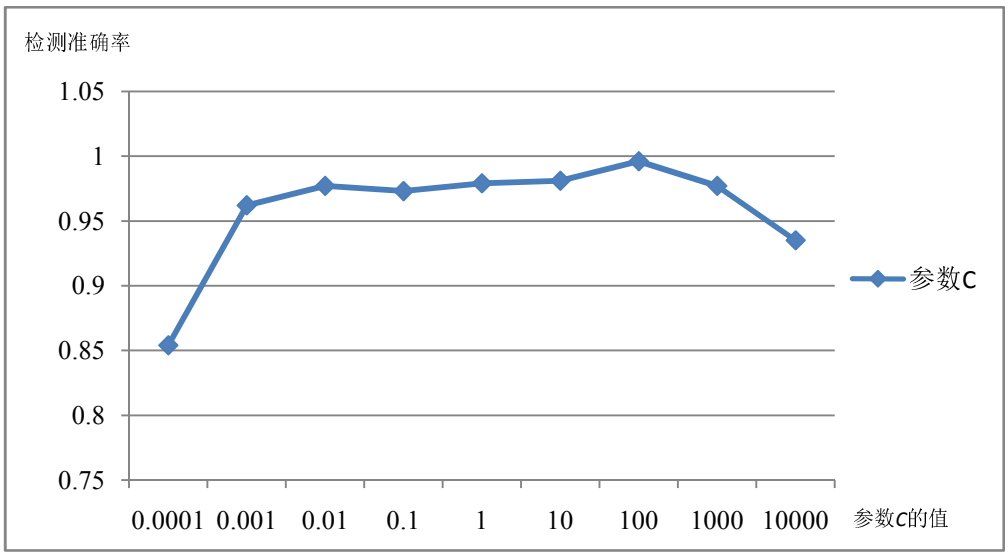

图 4.17　确定最佳参数 C

4.6.2　主动学习策略

1）主动学习策略流程

主动学习策略[42]通过询问机制对一些能够提升分类器性能的样本进行主动选择，然后使用这些样本重新设计分类器。主动学习策略能够以较少的训练样本数量实现较好的分类效果，也减少了标注样本所产生的人力和时间代价。主动学习策略的流程如图 4.18 所示。

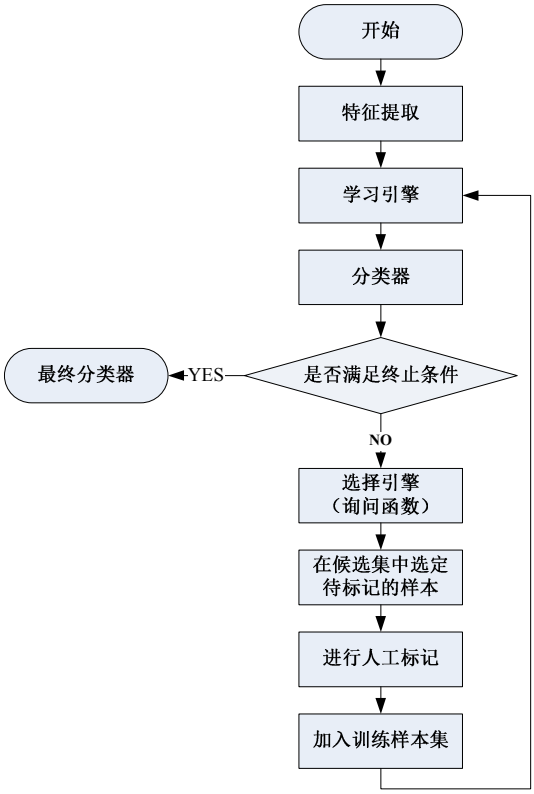

图 4.18　主动学习策略流程

2）询问函数

在图 4.18 中，询问函数是选择引擎的核心，它的主要功能是选择标记信息量最大且最有价值的样本，如何对这些样本进行选择是主动学习的关键。

对于样本集 $\phi=\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\},x_i\in R^n,i=1,2,...,n$，第 i 个样本集中第 i 个样本的特征向量用 x_i 进行表示，$y_i\in\{+1,-1\}$ 表示分类标识号，n 表示样本数量。

如果样本集是线性可分的，那么存在一个将超平面分类的决策方程：

$$w^Tx+b=0,x\in R^d \tag{4-15}$$

样本集中的所有样本 $(x_i,y_i)i=1,2,...,n$ 都满足下式：

$$\begin{cases} w^Tx+b\geq 1,y_i=1 \\ w^Tx+b\leq -1,y_i=-1 \end{cases} \tag{4-16}$$

那么包含于 R^n 的任意样本 $x=(x_1,x_2,...,x_n)^T$ 与分类平面之间的距离

$$d=\frac{w^T x_j+b}{\|w\|}，其中\|w\|=\sqrt{w^T\times w} \tag{4-17}$$

由公式（4-16）可得

$$d(x)=b^*+x\cdot w^*=\sum_{i=1}^{l} y_i\alpha_i^*(x\cdot x_i)+b^* \tag{4-18}$$

其中b^*表示分类阈值，b^*的值能够通过对两类中任一对支持向量取中值的方式得到。

我们对样本进行判断，针对待测样本x'来说，当$d(x')\le -1$时，代表x'被负样本包含；当$d(x')\ge 1$时，代表x'被正样本包含；当$1>d(x')>-1$时，代表x'被包含于超平面的分类间隔之内，此时，样本的属性难以确定，无法判断它是正样本还是负样本，另外，$d(x')$越趋近于 0，它的不确定性就越大。

由上述分析可知，当$\|d(x')\|<1$时，样本被包含于超平面的分类间隔当中，它的属性是不确定的，因此我们应该询问与分类超平面距离最近的样本点，对其进行人工标记，并将该样本添加到训练集中。

4.6.3 基于主动学习策略的SVM分类算法

1）框架描述

图 3.2 详细描述了基于主动学习策略的 SVM 分类模块的执行流程，该模块要求每一个 PDF 文件进行如下操作：①构造决策函数；②计算与 SVM 分类超平面之间的距离。主动学习策略中的询问函数可以对 PDF 样本的属性进行判断，当其属性不确定时，则认为该文件是“新型文件”，然后将它发送到安全专家处进行手工分析和标签操作。我们对“新型文件”定义如下：一类 PDF 样本文件，当它们被添加到训练集后，可以提升模型的预测能力和丰富反病毒软件的特征库。“新型文件”主要有两种形式：①位于分类超平面的隔离带中，该样本是恶意文件的可能性接近于它是正常文件的可能性；②位于超平面的恶意样本区域，由公式（4-17）进行计算，可知它与分类超平面之间的距离最远，如图 4.19 所示。在图中，左上方被圈住的的样本位于恶意样本区域，并且它们与分类超平面之间的距离最远，这类样本通常包含未知的恶意代码。接下来，这些“新型文件”被添加到训练集中，用于重新训练检测模型。此外，我们还需要将被标签为恶意文件的“新型文件”添加到反病毒软件特征库中。

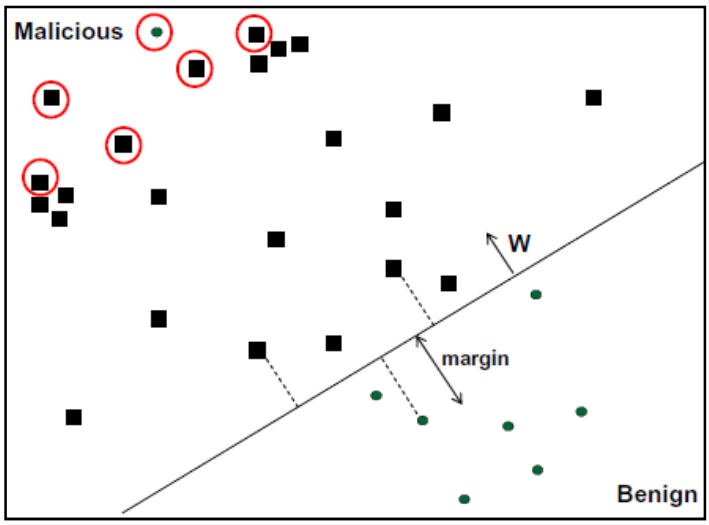

图 4.19　远离超平面的 PDF 恶意样本

基于主动学习策略的 SVM 分类模块的数据流图如图 4.20 所示。

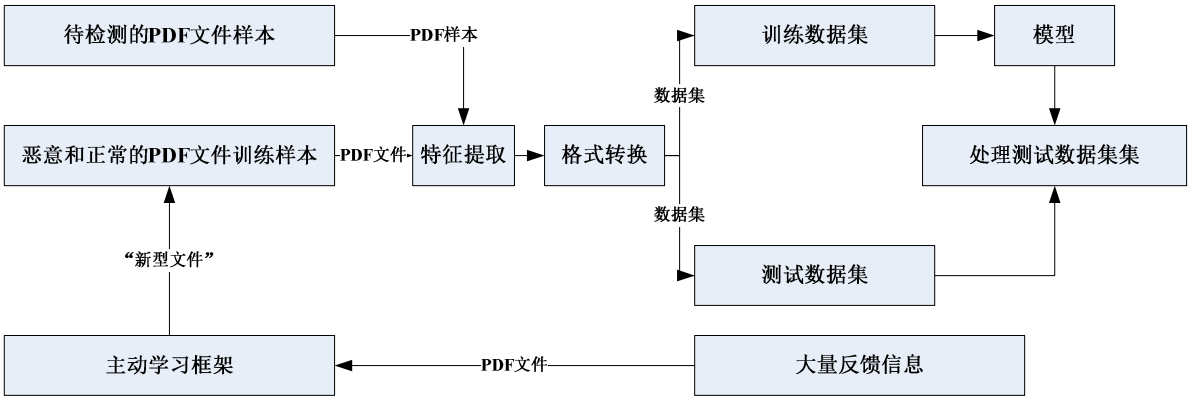

图 4.20　基于主动学习策略的 SVM 分类模块数据流图

2）训练和检测

基于主动学习策略的 SVM 分类模块集成了两个主要步骤：

①训练

检测模型是由一个包含恶意样本和正常样本的数据集进行训练的。

②检测和更新

基于主动学习策略的 PDF 文本检测子系统提供了分类和检测功能，它通过询问函数对“新型文件”进行识别。当安全专家完成标记“新型文件”的操作后，所有的“新型文件”将被训练集获取并进行训练，同时，恶意的“新型文件”被用于更新反病毒软件特征库。

主动学习策略是一种从未标记的样本中选择“新型样本”并对“新型样本”进行人工标记的技术，这个过程是迭代的。可以通过询问函数对“新型文件”进行选择，

当$\|d(x')\|<1$时，样本位于分类超平面的隔离带中，其属性（恶意还是正常）是难以确定的，我们对与分类超平面距离最近的样本进行询问，并通过人工的方式标记这些样本，然后将它们增加到训练样本集。相比于随机选择或被动学习方法，主动学习策略能够对一些提升检测性能的样本进行选择，减少了标记的工作量。

3）**样本分布测试**

本文 3.5.1 节描述了训练样本和预测样本的组成结构。预测样本的分布如图 4.21 所示。标记恶意样本为 1，标记正常样本为 0，样本点越趋近于 1 或 0，其属性（恶意还是正常）越容易确定，经分析可知，0.6 和 0.4 处的样本属性也是确定的，因此，在主动学习时选择 0.45 之后的 100 个样本加入训练集，对模型进行更新。

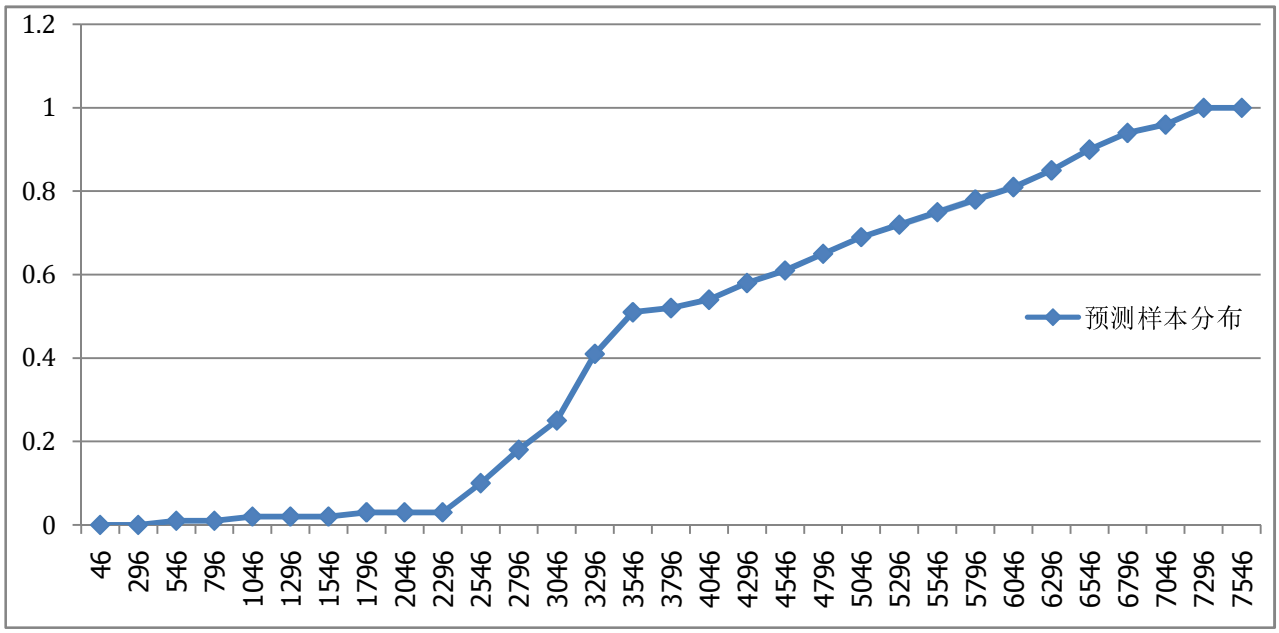

图 4.21　样本分布测试

4. 6. 4　测试结果分析

基于主动学习策略的 SVM 分类算法的实验步骤描述如下：

①选取测试样本和训练样本，其数量如表 4-3 所示。

②首先对 500 个样本进行训练，每一轮迭代后，训练样本数量增加 100。

③选择参数 C 的最优值，对剩下的样本进行测试。

④通过询问函数在训练模型中添加 100 个预测样本，重复步骤②和步骤③。

⑤迭代 10 次后，完成分类。

对基于主动学习策略的 SVM 分类算法和基于监督学习的 SVM 分类算法进行性能比较，分析训练样本数量逐渐增多时，两种算法检测准确率 Accuracy 的变化，比较结果如图 4.22 所示。

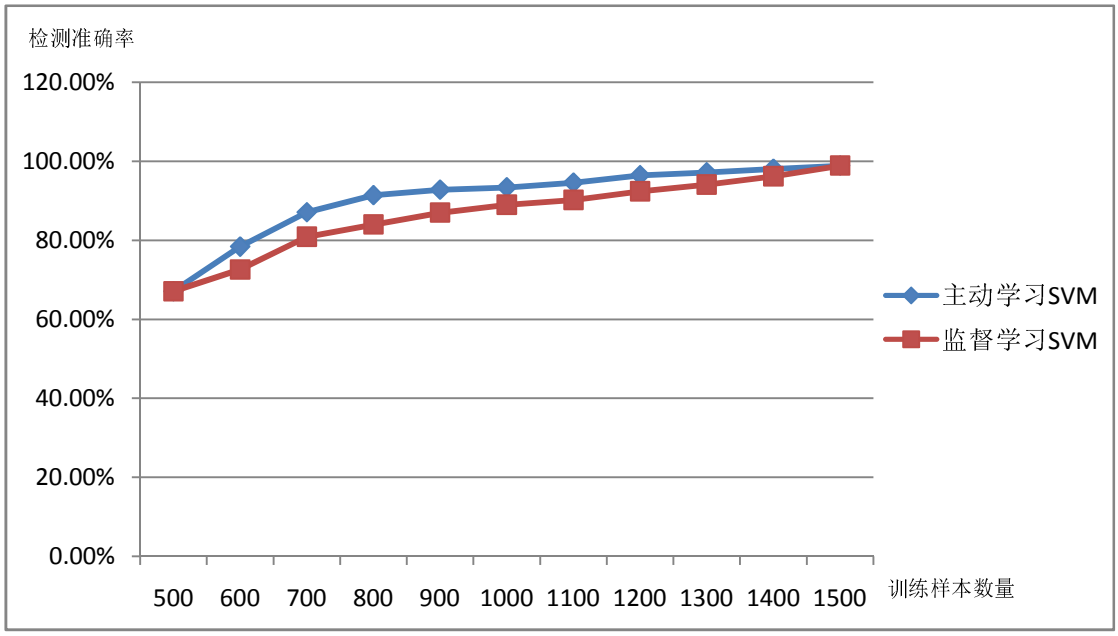

图 4.22　主动学习 SVM 与监督学习 SVM 检测结果比较

对图 4.22 进行分析，可以得到下述结论：

1）随着训练样本数量的增多，SVM 分类性能也在提升，这是因为选取的特征向量维数较高，特征的提取方法也比较合理、科学，在这种情况下，增加训练样本的数量就可以提升分类和检测效果。

2）基于主动学习策略的 SVM 分类方法具有更好的性能。起初，两种方法的检测准确率是接近的，之后基于主动学习策略的 SVM 分类方法性能提高得更快，这表明主动学习策略对传统 SVM 算法的性能提升有一定效果。

4.7　本章小结

本章介绍了基于主动学习策略的 PDF 文本检测子系统的总体架构及一些关键模块，包括 PDF 文件解析、Javascript 攻击检测、特征提取、特征选择和分类等，通过实验确定特征数量和 SVM 算法的最佳参数 *C*，比较基于主动学习的 SVM 分类算法与基于监督学习的 SVM 分类算法的测试结果。

第五章　PDF 漏洞检测子系统

5.1　系统总体架构

PDF 漏洞检测子系统主要包括 PDF 已知漏洞检测和漏洞利用关键代码检测两个核心模块，其总体架构如图 5.1 所示。PDF 已知漏洞检测模块涉及漏洞原理分析、漏洞检测规则库构建和基于特征匹配的静态检测三个流程，漏洞利用关键代码检测模块则主要针对 ROP 链。ROP 链本身是一些指令片段（gadget），用于突破 Windows 的内存防护机制 DEP，这些片段通常是以 ret 指令结尾的。对 ROP 链的检测主要采用堆栈检测、函数调用检测、函数参数检测和动态链接库检测四种方法。

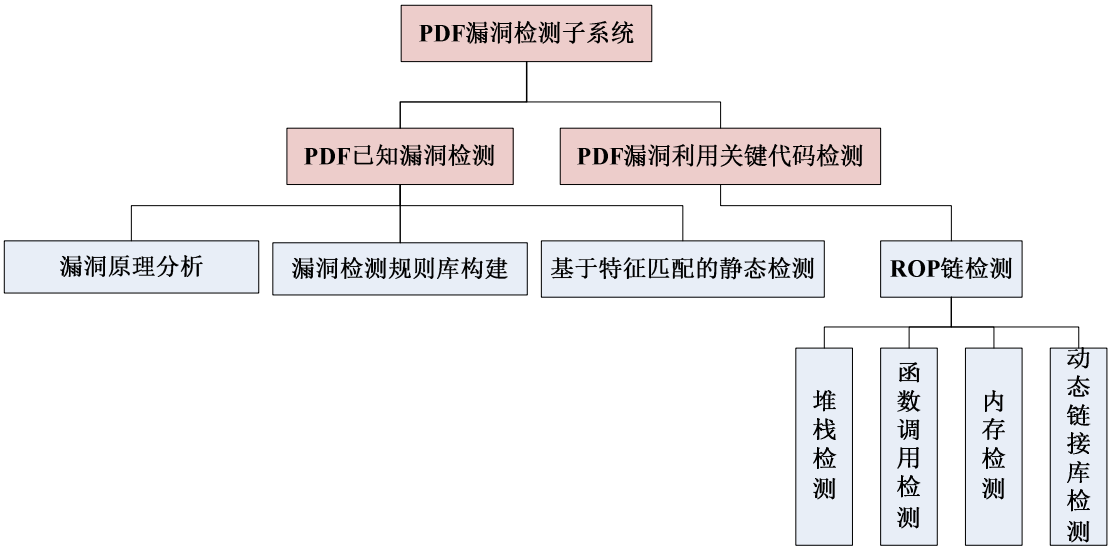

图 5.1　漏洞检测子系统总体架构

5.2　PDF 已知漏洞检测

5.2.1　PDF文件格式漏洞简介

近年来，PDF 文件以其便携性和易用性受到了人们的广泛关注，因此，针对该种类型文件的安全漏洞具有较大危害性。PDF 文件格式漏洞[21]以缓冲区溢出为主。缓冲区溢出是一种常见的攻击方式，它向较小的缓冲区中写入较大数据，从而覆盖堆栈的原有信息，并对内存中数据的值进行修改。这样，之前的程序执行流程很有可能会发生改变，使得恶意代码被执行，最终完成对用户主机的控制。

PDF 缓冲区溢出漏洞发生的原因是程序员在编写代码时忽略了对缓冲区的长度检查或者假定分配的内存空间超过了数据的长度。PDF 缓冲区溢出漏洞可以分为两种：

堆溢出和栈溢出。堆喷射技术是堆溢出漏洞的一种利用方法，它可以将程序的执行流程转移到嵌入的恶意 Javascript 代码上，堆喷射技术的具体细节见本文 2.2.1 节。

5.2.2 PDF漏洞分析

1）漏洞分析技术

漏洞分析技术指的是对已公开的安全漏洞进行原理分析，通过 POC 代码触发漏洞，对漏洞场景予以重现并编写漏洞检测规则等。为完成漏洞分析过程，我们需要在代码中定位漏洞，理清攻击的基本原理，并对漏洞的潜在利用方法进行估计，因此，漏洞分析是一项具有较高挑战性的工作。

漏洞分析技术通常包含信息采集、分析调试以及漏洞利用分析三个阶段，其流程如图 5.2 所示。漏洞分析技术为修复漏洞和构建漏洞检测规则库提供了支持。重现漏洞是漏洞分析的重要步骤，只有找到合适的触发条件、触发步骤及受影响软件的版本后，才能稳定地重现漏洞，然后采用调试和跟踪的方法对漏洞进行分析。为重现漏洞，通常需要对 POC 代码进行编写。

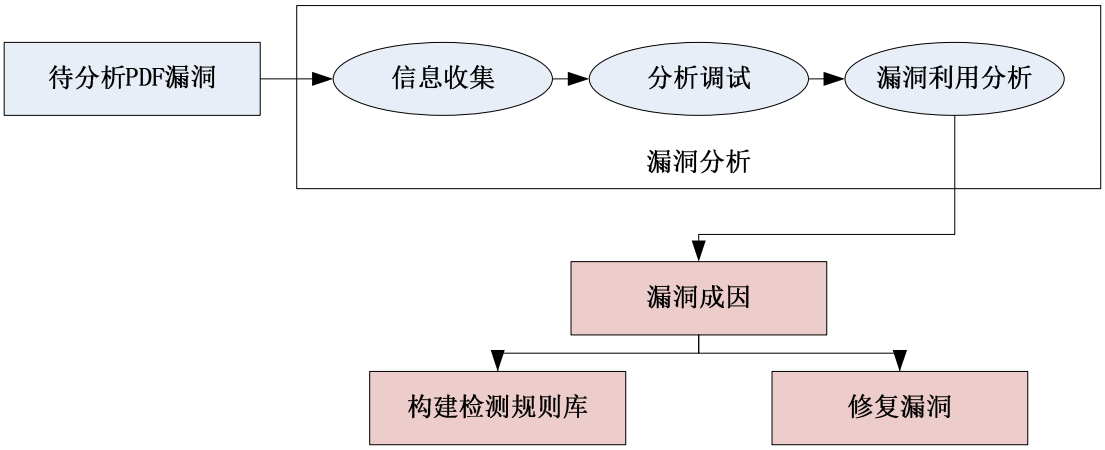

图 5.2　漏洞分析流程

①信息收集

对信息的收集是漏洞分析过程的第一个阶段，它主要是在分析调试之前，收集漏洞的相关信息，包括漏洞基本信息、文件格式信息和系统状态信息等。

a. 漏洞基本信息

对国内外著名的软件厂商网站和漏洞库进行检索，分析由它们发布的漏洞公告，获得漏洞名称、CVE 编号、漏洞类型、受影响的产品、漏洞公布时间和漏洞严重等级等漏洞公告信息。

以 PDF 漏洞 CVE-2011-2437[43]为例，对其漏洞公告进行收集，可知该漏洞的名称是 Adobe Acrobat and Reader CVE-2011-2437 Remote Heap Buffer Overflow Vulnerability，漏洞类型是远程可利用，受影响的产品是 Windows 平台下的 Adobe Reader 8.x（8.3 之

前版本）和 9.x（9.3.5 之前版本），漏洞公布时间是 2011 年 9 月 13 日，漏洞的严重等级是高危。

b. 文件格式信息

对存在漏洞的应用程序的文件格式进行解析，从而在漏洞分析过程中能够更快地对诱发漏洞的函数、参数和数据对象进行定位。

c. 系统状态信息

观察漏洞触发时的系统状态，对寄存器数据、堆栈数据以及进程、线程的上下文状态进行监测。

②分析调试

漏洞分析的核心步骤是分析调试，通过信息收集，对触发漏洞的特定数据或字段进行记录，找到它们和漏洞点之间的对应关系。另外，需要采用调试工具对异常信息进行监控，动态跟踪程序在解析特定文件格式时的系统状态，通过回溯法对漏洞的成因和机理进行分析。

a. 分析调试工具

在对漏洞进行分析调试时，工具是很重要的，常见的漏洞分析工具有：虚拟机软件 VMware、文本编辑工具 UltraEdit、静态分析工具 IDA 和动态调试工具 OllyDbg、WinDbg、Immunity Debugger 等。动态调试工具 OllyDbg 仅可以调试 Ring3 级的应用程序，它具有丰富的插件和很强的可扩展性，提高了工具的灵活性；而 WinDbg 对 Ring0 级的内核程序和 Ring3 级的应用程序都提供了支持。

b. 跟踪和监测数据流

数据流指的是程序在运行过程中从输入到输出的一条执行路径，这个路径上的节点覆盖和修改寄存器、内存的方式是我们应该关注的，对寄存器和内存的不当处理和非法操作往往是安全漏洞产生的直接原因。对数据流的跟踪和监测可以采用捕获异常和设置断点两种方法。

捕获异常：通过动态调试工具能够对异常事件进行捕获，当正在运行的程序出现异常时，它的执行将被挂起。如当 Windbg 捕获到由缓冲区溢出导致的异常时，该工具会通过 KiUserExceptionDispatcher（一种异常分发函数）的参数信息，对异常出现时堆栈中的数据长度和参数位置等信息进行观察。

断点设置：断点包括普通断点、硬件断点和内存断点等类型。内存访问断点是内存断点的一种，它可以在特定内存的数据发生改变时将程序挂起，此时，就可以分析程序的上下文了，WinDbg、OllyDbg 和 Immunity Debugger 三种动态调试工具都可以对内存断点进行设置。

c. 漏洞原理分析

当漏洞稳定触发后，通过跟踪和监控异常信息，可以得到漏洞点代码（即程序中

触发漏洞的代码段），漏洞点代码通常位于一个函数或方法中，此时，可以结合调用栈信息，采用回溯分析的方法对漏洞原理进行分析，最终在样本的文件格式中找到诱发漏洞的数据对象。

在对漏洞原理进行分析的过程中，可以将动态调试工具和静态分析工具结合起来，从而提高分析效率。通过动态调试工具对存在漏洞的程序进行加载，然后跟踪程序的执行过程，可以采用单步执行的方式依次执行每一条汇编语句，观察堆栈、寄存器和内存中数据的变化，也可以通过回溯法定位产生溢出的漏洞函数，从而快速剖析漏洞原理。IDA 等静态分析工具能够反汇编存在漏洞的程序，并获得程序的总体结构和完整的反汇编代码，通过对反汇编代码进行阅读和分析，可以进一步理清代码功能，找到代码中的缺陷。静态分析工具主要是用于辅助动态调试工具的。

③漏洞利用分析

漏洞利用分析是在漏洞原理分析的基础上，结合漏洞触发条件，对漏洞的危害程度和可利用程度进行分析，并建立漏洞检测规则的过程。

漏洞利用分析首先需要对漏洞的类型（包括本地权限提升、缓冲区溢出或任意代码执行等）进行确定，然后结合触发漏洞的条件分析漏洞的利用条件，确定是否是特定地址读写或任意地址读写等，最后对漏洞可能造成的危害进行预测。

2）PDF 漏洞分析实例

通过收集 PDF 漏洞 CVE-2011-2437 的漏洞公告信息，可知该漏洞产生的原因是 Adobe Reader 和 Adobe Acrobat 在运行过程中出现了堆溢出，下面对 CVE-2011-2437 漏洞的分析调试过程和检测方法进行介绍。

① CVE-2011-2437 漏洞触发

在 WinDbg 中运行 AcroRd32.exe（版本号 9.3.3.177），漏洞触发时的上下文状态如图 5.3 所示：

```
0:000> g
(508.554): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=12b7eff0 ebx=00000002 ecx=00000008 edx=12ad6ff8 esi=0012d6bc edi=00000002
eip=130f586b esp=0012d69c ebp=00000000 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
2d!E3DLLFunc+0x3887:
130f586b 890411          mov     dword ptr [ecx+edx],eax ds:0023:12ad7000=????????
```

图 5.3　漏洞触发时的上下文状态

② CVE-2011-2437 调试分析

首先，我们对处理 PCX 图像的反汇编代码进行定位，这段代码用于从 POC 文件中读取数据，将读取到的数据赋值给 Xmin、Ymin、Xmax 和 Ymax 四个图像边界变量，并在寄存器 ecx 中存储计算得到的缓冲区大小。通过 recx 命令对 ecx 的值进行查看，

可知 ecx=0x00010000。

```
0:000>uf 130f5f1b
2d!E3DLLFunc+0x3f37:
130f5f1b   0fb74dec          movzx   ecx,word ptr [ebp-14h]
130f5f1f   0fb745f0          movzx   eax,word ptr [ebp-10h]
130f5f23   0fb755ee          movzx   edx,word ptr [ebp-12h]
130f5f27   2bc1              sub   eax,ecx
130f5f29   0fb74df2          movzx   ecx,word ptr [ebp-0Eh]
130f5f2d   2bca              sub   ecx,edx
130f5f2f   660fb655eb        movzx dx,byte ptr [ebp-15h]
130f5f34   40                inc   eax
130f5f35   41                inc   ecx
130f5f36   807d2901          cmp   byte ptr [ebp+29h],1
130f5f3a   898562ffffff      mov   dword ptr [ebp-9Eh],eax
130f5f40   898d66ffffff      mov   dword ptr [ebp-9Ah]
0:000> r ecx
ecx=00010000
```

读取缓冲区大小的代码段如图 5.4 所示：

```
130f581c 0fb7460a        movzx   eax,word ptr [esi+0Ah]   ds:0023:0012d6c6=0000
130f5820 33c9            xor     ecx,ecx
130f5822 6a04            push    4
130f5824 5a              pop     edx
130f5825 f7e2            mul     eax,edx
130f5827 0f90c1          seto    cl
130f582a f7d9            neg     ecx
130f582c 0bc8            or      ecx,eax
130f582e 51              push    ecx
130f582f e88a5c0400      call    2d!e3_EXTENSION::IsTypeOf+0x19 (1313b4be)
```

图 5.4　读取缓冲区大小

接下来对 esi+0Ah 处存储的内容进行查看，如下所示：

```
0:000>dd   esi+0ah
0012d6c6   00010000 00180001 00000000 00000000
0012d6d6   00000000 00000000 00000000 00000000
0012d6e6   d7600000 00060012 8fb00000 9f90151c
0012d6f6   d6a412fd 5f020012 d898130f 00c20012
0012d706   323011e8 00005643 00000000 00000000
0012d716   00000000 00000000 00000000 00000000
0012d726   00000000 00000000 00000000 00000000
0012d736   00000000 00000000 d8600000 4e700012
```

由 dd 命令执行结果可知，esi+0Ah 处存储的内容即为缓冲区大小 0x00010000。分析图 5.4 中的关键汇编代码指令 movzx eax,word ptr [esi+0Ah]，由于 word 表示两个字节，因此，该指令只读取了 esi+0Ah 处存储数据的第 16 位，即 0x0000，此时读取到的缓冲区大小为 0x0000。继续执行程序，直到程序发生崩溃，崩溃的原因是 ecx 的值超

过了缓冲区的大小。

```
0:000> g
(350.348): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=12b7eff0 ebx=00000002 ecx=00000008 edx=12ad6ff8 esi=0012d6bc edi=00000002
eip=130f586b esp=0012d69c ebp=00000000 iopl=0         nv up eiplnznaponc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
2d!E3DLLFunc+0x3887:
130f586b 890411          movdwordptr [ecx+edx],eax ds:0023:12ad7000=????????
```

经过调试可知，缓冲区大小是由 YMax-YMin+1 计算得到的。使用 010 Editor 对 POC 文件中的 PCX 对象进行分析，可以发现 Xmin、Ymin、Xmax 和 Ymax 四个变量在内存中分别占用两个字节，共占用八个字节，如图 5.5 所示：

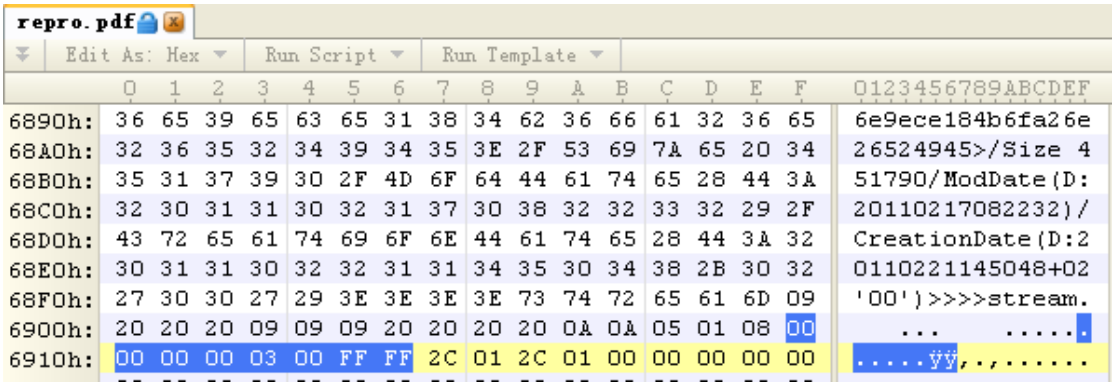

图 5.5　PCX 对象内容分析

图 5.5 中标黑的部分由左至右依次为：XMin=0000，YMin=0000，XMax=0003，YMax=ffff，对缓冲区大小进行计算： buffer=YMax-YMin+1=10000，取 buffer 的第 16 位则会得到 0x0000，从而触发漏洞。

③ CVE-2011-2437 漏洞原理

该漏洞产生的原因是 Acord32.exe 在执行 2d 模块时，会对 PCX 图像对象的头部进行处理，并分配一个 32 位的堆缓冲区，但是在对该堆缓冲区进行使用时只读取了其低 16 位数据，从而触发了堆缓冲区溢出漏洞。

④ CVE-2011-2437 漏洞检测规则

为对 CVE-2011-2437 漏洞进行检测，首先定位 PCX 对象数据流的起始位置，然后计算 YMax-YMin+1 的值，其中，YMax 偏移数据流起始位置 0x1C 个字节，是 word 类型，YMin 偏移数据流起始位置 0x18 个字节，也是 word 类型。如果 YMax-YMin+1 的计算结果大于 0xFFFF，那么可能会触发漏洞。

5.2.3　PDF漏洞检测规则库构建

通过对 PDF 漏洞的触发条件和原理进行分析，可以抽取到漏洞特征，进而形成针对单个 PDF 漏洞的检测规则。将 19 个 PDF 已知漏洞的检测规则组合起来，可以形成一个检测规则库，如表 5.1 所示。PDF 漏洞检测规则库包含 CVE 编号、受影响的软件和版本以及漏洞检测规则等信息。

表 5.1　PDF 漏洞检测规则库

CVE 编号	受影响的软件和版本	漏洞检测规则
CVE-2011-2096	Adobe Reader 及 Acrobat: 8.38.x 9.x（9.4.5 之前）	检测 Universal 3D 字段，如果文件头中定义的数据长度小于 TextureName 的长度+4，将会触发漏洞
CVE-2011-2097	Adobe Reader 及 Acrobat: 8.x（8.3 之前） 9.x（9.4.5 之前） 10.x（10.1 之前）	检测网络数据流，当发现 PDF 文件中存在“/ICCBased”字段时，对 ProfileDescriptionTag 字段（‘desc’）进行查找。若该字段存在，则判断其后偏移 22 字节处的值是否大于 0x7FFFFFFF，若是则触发漏洞
CVE-2011-2098	Adobe Reader 及 Acrobat: 8.x（8.3 之前） 9.x（9.4.5 之前） 10.x（10.1 之前）	检测网络数据流，当发现 PDF 文件中存在“/JPXDecode”字段时，对 JP2C box 字段（‘jp2c’）进行查找。若该字段存在，则判断其后偏移 11 字节处的值是否大于 0x3f，若是则触发漏洞
CVE-2011-2101	Adobe Reader 及 Acrobat: 10.0.x 8.x（8.3 之前） 9.x（9.3.5 之前）	在间接对象的 dictionary 中查找字符串“/S /GoToR”，检查“/S/ GoToR”所在的 dictionary 中是否存在“/F”，如果“/F”入口的值是一个 Javascript URL ，则会触发漏洞
CVE-2011-2105	Adobe Reader 及 Acrobat: 10.0.x 8.x（8.3 之前） 9.x（9.4.5 之前）	在 PDF 文件中查找“/Subtype /CIDFontType0”或“/Subtype /CIDFontType0”标记，然后检测所有“/w”标记，当出现“/W [0 [778 0] 2 3 250 4 [333 408] 6 7 500 8 [833 778 180]”等数据段时，触发漏洞
CVE-2011-2432	Adobe Acrobat 9.x Adobe Reader 9.x	解析 PDF 文件中的 TIFF 资源，在 TIFF::IFDEntries::ImageFileDirectory::FieldEntries 列表中寻找 NumberOfValues 字段，若 NumberOfValues 字段的值和其类型长度的乘积大于 8，则会触发漏洞
CVE-2011-2433	Adobe Reader 及 Acrobat: 10.0.x 8.x（8.3 之前）	在偏移 PICT 图像流 0x5C 和 0x5E 的位置得到图像的宽度和高度，计算缓冲区的大小（宽度*高度*4 +28 字节），如果该值小于 RLE 加密

	9.x（9.4.5 之前）	包的大小，则会触发漏洞
CVE-2011-2434	Adobe Acrobat 9.x Adobe Reader 9.x	在 PDF 文件流中查找四字节字符串“PICT”，如果偏移该位置 0xF 处是 0x2E0，则会触发漏洞
CVE-2011-2435	Adobe Reader 及 Acrobat： 10.0.x 8.x（8.3 之前） 9.x（9.3.5 之前）	在偏移 PDF 文件头 0x58 的位置找到四字节字符串“PICT”，在偏移解码流数据 0x68 的位置找到第一个数据包，如果数据包的第一个字节不是 0x01，那么这个数据包不是最后一个数据包，接下来按同样的方式对第二个数据包进行检测，在 PDF 样本中，如果数据包的数量超过 8，则会触发漏洞
CVE-2011-2436	Adobe Reader 及 Acrobat： 10.0.x 8.x（8.3 之前） 9.x（9.3.5 之前）	找到包含 IFF 图像的数据流，解析 IFF 图像的 TBHD 和 RGBA 数据块，从而得到 TBHD 数据块的宽度（偏移 TBHD 数据块 0x05 字节，unit32 类型）和 RGBA 数据块的 X2（偏移 RGBA 数据块 0x9 字节，unit16 类型）的值。若 X2 的值大于或等于 Width 的值，则会触发漏洞
CVE-2011-2437	Adobe Reader 及 Acrobat： 10.0.x 8.x（8.3 之前） 9.x（9.3.5 之前）	首先定位 PCX 对象数据流的起始位置，然后计算 YMax-YMin+1 的值，其中，YMax 偏移数据流起始位置 0x1C 个字节，是 word 类型，YMin 偏移数据流起始位置 0x18 个字节，也是 word 类型。如果 YMax-YMin+1 的计算结果大于 0xFFFF，那么可能会触发漏洞
CVE-2011-2438	Adobe Reader 及 Acrobat： 10.0.x 8.x（8.3 之前） 9.x（9.4.5 之前）	在 PDF 文件中识别 bitmaps，判断 4 * ColorsUsed > (width*(bitsPixel & 0x0ffffff) + 31) / 32) * 4) * height + 4 * (2 * BitCount)是否成立，如果成立，则触发漏洞
CVE-2011-2440	Adobe Reader 及 Acrobat： 10.0.x 8.x（8.3 之前） 9.x（9.4.5 之前）	首先定位 JFIF 的数据流，然后在其数据流中搜索 0xFFE0，如果 0xFFE0 的个数多于 1，则会触发漏洞
CVE-2011-2462	Adobe Reader（9.4.6 之前） Adobe Acrobat X（10.1.1 之前）	若 PDF 文件数据结构中的 shader_list_count 的值不为 1，则会触发漏洞
CVE-2011-4369	Adobe Reader 及 Acrobat： 9.x 10.x	查找 MarkupLinkedItem 中的 ExtendedEntityReference->ContentEntityReference->ReferenceData 结构，对于每一个结构，若其中没有 PRC_UID，则会触发漏洞
CVE-2012-4149	Adobe Reader 及 Acrobat：	如果 PDF 文件中包含一个 Annot[0]类型的流，

	9.x（9.5.2 之前） 10.x（10.1.4 之前）	获得子类型入口，如果子类型是 Free 文本，获得/IT 的值，如果/IT 的值不为 FreeText、FreeTextTypewriter 和 FreeTextCallout 之一，则会触发漏洞
CVE-2012-4155	Adobe Reader 及 Acrobat： 9.x（9.5.2 之前） 10.x（10.1.4 之前）	如果 PDF 文件中包含一个 Type1C[0]类型的数据流，那么可以获得“Top Dict INDEX”[1]和“Charstrings”入口（“Charstring INDEX”的偏移地址），通过“Charstrings”入口可以获得“Charstring INDEX”，其中的内容若不是有序的，则会触发漏洞
CVE-2012-4157	Adobe Reader 及 Acrobat： 9.x（9.5.2 之前） 10.x（10.1.4 之前）	如果 PDF 文件中包含一个 Font format[0]类型的流，那么可以获得 cmap 表的偏移量。当 numberSubtables 大于 0 时，会对每一个 sub-table 文件头进行解析。如果 sub-table 的格式是 0xC，且其长度小于 0x10 时，则会触发漏洞
CVE-2012-4159	Adobe Acrobat 9.x（9.5.2 之前） Adobe Acrobat 10.x（10.1.4 之前）	如果 PDF 文件中包含一个 subtypeCIDFontType0C [0]类型的流，那么可以获得“Top Dict INDEX” 和“Charstrings”入口（“Charstring INDEX”的偏移地址），通过“Charstrings”入口可以获得“Charstring INDEX”，对于每一个 charstring 流，如果检测到“cntrmask”标记，则说明其存在漏洞

5.2.4　基于规则匹配的静态检测

基于规则匹配的静态检测方法是将表 5.1 中描述的检测规则与待查 PDF 样本进行匹配，如果 PDF 样本中有一个或多个数据结构的值命中了规则库中的一条或多条检测规则，那么就判定该样本文件存在 PDF 已知漏洞。

基于规则匹配的静态检测方法是以PDF文件解析为前提的，只有将PDF文件中数据结构的类型、名称、偏移量、值和长度等属性准确提取出来，才可以进行下一步的逻辑判断。与PDF漏洞相关的数据结构主要有BMP、IFF、TIFF、PCX、PICT、Universal3D、JPEG和TrueTypeFont等，因此，我们在解析PDF文件本身时，也需要对这些数据结构进行格式解析。

基于规则匹配的静态检测方法流程如图 5.6 所示。

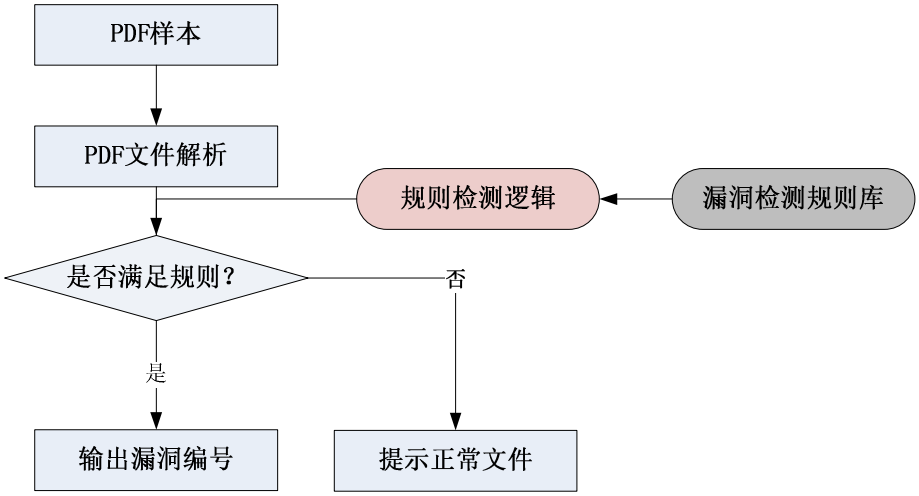

图 5.6　基于规则匹配的静态检测方法流程

本文 5.2.2 节对 PDF 已知漏洞 CVE-2011-2437 的调试过程、原理和检测规则等进行介绍。由分析可知，当 YMax-YMin+1 的值大于 0xFFFF 时，就会触发漏洞，将 CVE-2011-2437 的检测规则转换为 C#语言编写的代码，如下所示：

```
using System;
using System.Collections.Generic;
using GUT.Architecture;
using GUT.DataFormats.PCX;
namespace GUT.DataFormats.PDF2{
    public partial class PDFDetectionLogic : PDF2{
        [CanDetect("CVE_2011_2437")]
        public void DetectCVE_2011_2437(){
            List<PCXHeader> hdrs = this.FindSubDataStructuresOfType<PCXHeader>(true);
            foreach (PCXHeader hdr in hdrs){
                if ((UInt32)hdr.YMax.Value - (UInt32)hdr.YMin.Value + 1 > 0xffff){
                    hdr.AddParsingNote(ParsingNoteRegular.PossiblyMalicious, "PCX
                    YMax - YMin + 1: Overflows 16bit value", "CVE_2011_2437");
                    return;
                }
            }
        }
    }
}
```

PDF 已知漏洞检测模块可以对表 5.1 中描述的 19 种 PDF 已知漏洞进行检测，每一种漏洞分别对应一段规则检测代码。

5.3　PDF 漏洞利用关键代码检测

5.3.1　ROP技术简介

DEP 防护机制[44]通过阻止指令在数据区（包括堆、栈、内存池等）上执行，提升了 PDF 漏洞利用的难度。当前，ROP 技术已成为突破 DEP 的最有效方法。ROP 的基本原理是：在栈上对一些指令组合的返回地址进行布置，从而达到执行代码的目的，这些指令组合通常以 ret 指令结尾，这样才能将它们依次执行。尽管指令无法在栈上执行，但可以利用可执行数据区的指令组合完成特定功能。

图 5.7 描述了由 ROP 链构成的 Shellcode。图中左半部分表示堆、栈等数据区，右半部分表示可执行代码区。指向可执行代码区的地址构成了数据区中的 Shellcode。这些地址指向的代码片段有一个共同的特点，即都以 ret 指令结尾。这样能够使程序沿着数据区的地址程序，可以有效收回控制权。

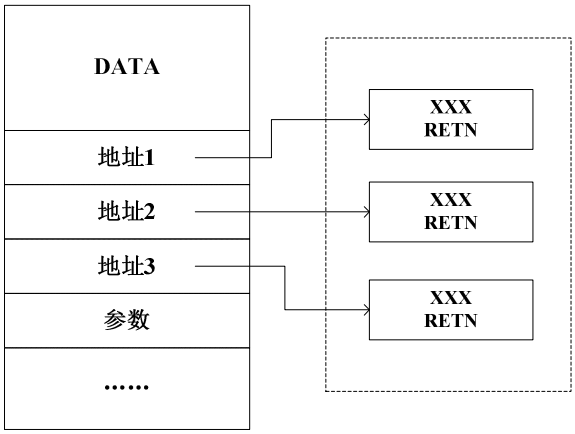

图 5.7　ROP 链构成的 Shellcode

典型 ROP 链的特征是：

1）当敏感 API 函数返回后，下一个代码片段会执行，这个代码片段一般不是由 call 指令调用的；

2）每个代码片段包含的指令数量很少，当代码片段执行完毕后，会 return 到下一个敏感 API 函数或下一个代码片段。

通常，单纯通过 ROP 链来实现一个正常 Shellcode 的全部功能是很困难的。因此，在执行 ROP 链之前应先调用系统 API 函数来对内存属性进行修改或分配一段可执行内存，常见的敏感 API 函数[44]有：

- ✧ VirtualProtect/VirtualProtectEx：使 Shellcode 所在的堆、栈等区域可执行；
- ✧ VirtualAlloc/VirtualAllocEx：对具有可执行权限的内存进行分配，然后复制 Shellcode 到这段内存中执行；
- ✧ zwSetInformationProcess：修改 KPROCESS 结构的_KEXECUTE_OPTIONS 中

的 DEP 设置，将 DEP 关闭；

- ✧ SetProcessDEPPolicy：为 DEP 设置不同的模式；
- ✧ LoadLibraryA/LoadLibraryW/LoadLibraryEx/LoadLibraryExW：加载动态链接库；
- ✧ HeapAlloc：在指定的堆上分配内存。

5.3.2 ROP链检测方法

ROP 链检测模块会对上文描述的敏感 API 函数进行 inline-hook，它能够直接修改敏感 API 函数当中的指令，以一个跳转或其它指令来完成挂钩，使敏感 API 函数在执行之前进行额外的验证。当敏感 API 函数被调用后，会先跳转至 ROP 链检测模块的钩子函数上进行检测，之后再恢复程序的正常执行流程。

对敏感 API 函数挂钩的操作是通过 Detours[45]实现的，Detours 是 Microsoft 研发的一个函数库，它可以将任意数据段插入到 PE 文件中，对 dll 文件的导入表进行修改，也可以拦截 X86 主机上的任意 Win32 API 函数。Detours 是在汇编层进行处理的，它能够对目标 API 函数出口和入口处的汇编指令进行修改。

ROP 链检测模块流程如图 5.8 所示：

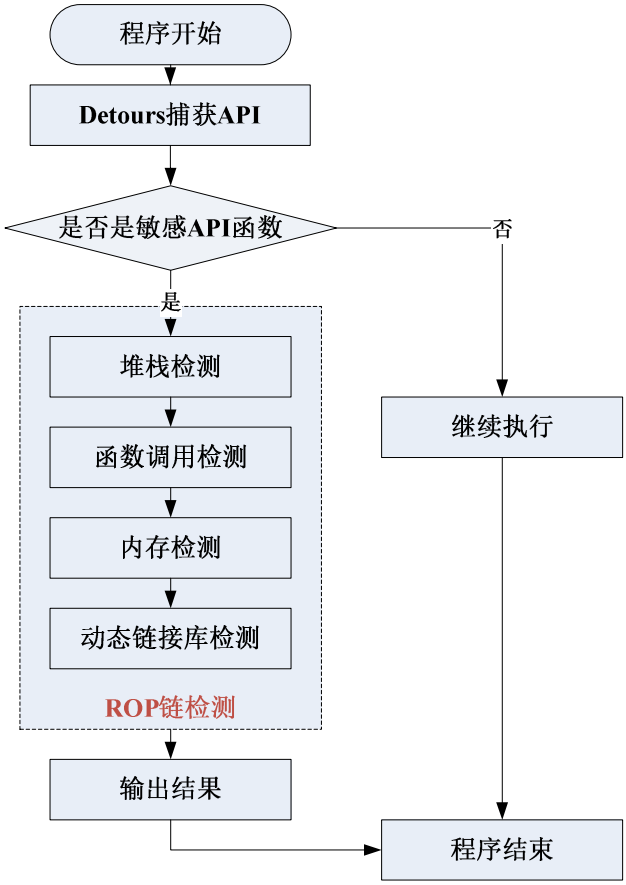

图 5.8　ROP 链检测模块流程

ROP 链检测有四种方法，分别为堆栈检测、函数调用检测内存检测和动态链接库检测等，下面对四种方法进行分析：

1）堆栈检测

当前一种常见的实现 ROP 技术的方法是将用户控制的寄存器（该寄存器可能指向用户控制的地址）和栈指针互换，实现该过程的代码片段如下所示：

```
XCHG　EAX, ESP
RET
```

对这类攻击的检测方法是当调用 VirtualProtect、VirtualAlloc 等敏感 API 函数时，检测栈指针 ESP 是否位于当前线程栈空间范围，如果不位于该范围，则抛出异常。

堆栈检测方法只是检测当敏感 API 函数被调用时，ESP 指针是否位于栈区域当中，并没有充分考虑 ROP 链的根本特征，因此 ROP 的简单变形就可以绕过该检测。为绕过堆栈检测方法，只需通过 ROP 链在栈区布置 API 参数，并将 ESP 重新指向栈区，其关键代码如下所示：

```
pop eax; retn
param-> eax              //参数
mov [esi],eax; retn      //esi 指向栈顶
dec esi; retn             //esi减1，上移
dec esi; retn
dec esi; retn
dec esi; retn
```

上述指令片段可以将一个参数写入堆栈。pop eax 表示向 eax 中弹入一个参数。mov [esi],eax 将参数写入 esi，esi 此时指向栈顶。之后的 4 个 dec esi 用于将栈顶抬高，为下一次写入参数作准备。上述指令片段不仅可以写入参数到堆栈，还可以将返回地址写入堆栈。当参数与返回地址都布置在堆栈上之后，可以将要调用的敏感 API 函数写入堆栈，然后使用 xchg esi,esp 对当前的栈顶指针 esp 进行修改。这样就会使函数调用时，ESP 指针处于合法范围内，从而绕过对 ROP 链的堆栈检测。

2）函数调用检测

函数调用检测方法对敏感 API 函数的返回地址进行验证，如果返回地址的上一条指令是 call，且 call 指令的跳转目标是敏感 API 函数的地址时，那么通过验证。如果返回地址的上一条指令不是 call，或 call 指令的目标跳转地址不是敏感 API 函数的地址，则认为检测到 ROP 链。

函数调用检测方法用于过滤敏感 API 函数的调用来源，确保目标地址是通过 call 指令进入的，而不是通过 jmp 或 return 指令进入的。call 指令的跳转方式主要有以下五种：①call [reg+disp32], call [loc32]；②call rel；③call reg, call [reg]；④call [reg+disp8]；⑤call [reg1+reg2+disp32]。

3）内存检测

内存检测方法主要针对栈溢出漏洞利用的 ROP 链进行检测。它通过验证传入到敏感 API 函数 VirtualProtect/VirtualProtectEx 中的参数，对 ROP 链进行判断。VirtualProtect/VirtualProtectEx 函数用于对内存页的访问属性进行修改，它的参数有 lpAddress、dwSize 和 flwNewProtect。

内存检测方法首先对第三个参数 flwNewProtect 进行检查，如果发现参数 flwNewProtect 中设置了可执行标记位，那么就对参数 lpAddress 进行检查。如果内存[lpAddress,lpaddress+dwSize]位于当前线程的栈空间范围内，即下式成立：

TEB.StackTop < lpAddress && (lpAddress+dwSize) < TEB.StackBottom （5.1）

那么就认为检测到了 ROP 链。

4）动态链接库检测

动态链接库检测方法主要为了防止黑客通过执行 ROP 链从远端服务器加载恶意的 dll 文件。该种检测方法对 LoadLibraryW/LoadLibraryA/ /LoadLibraryEx/LoadLibraryExW 的参数进行检验，其关键代码如下所示：

```
if(lpFileName[0] == '\\' && lpFileName[1] != '?'){    //检测到从远端服务器加载dll文件
    return FALSE;
}
else{
     return TRUE;
}
```

5.4 本章小结

本章首先对 PDF 漏洞检测子系统的总体架构进行介绍，然后结合 PDF 漏洞分析实例，对漏洞检测规则库进行构建，提出一种基于规则匹配的 PDF 已知漏洞检测方法，接下来描述 ROP 技术的原理，分析 ROP 链检测的若干方法。

第六章　PDFCheck 模型实现与测试

6.1　开发环境简介

PDFCheck 模型以 C#作为编程语言，采用.Net Framwork 框架，其开发平台选用 Visual Studio 2010，下面分别对其进行介绍：

6.1.1　C#语言

C#是一种面向对象的编程语言。C#源代码被编译后会生成中间字节码。C#继承了 Java、C、C++和 VB 语言的一些特性，它主要用于对.NET 环境下的应用程序进行开发。

C#使 C++程序员能够高效地开发程序，它可以对由 C/C++编写的本机原生函数进行调用，因此不会削弱 C/C++原有的丰富功能。C#与 C/C++极其相似，这就使得熟悉 C++的程序员可以快速地转向 C#。C#的语法简单、灵活，它对 C++的一些复杂语法特性进行简化，并提供了委托、枚举、空值类型和直接内存访问等功能。同时 C#对泛型方法和类型提供支持，从而提供更多的安全特性。C#语言的环境架构如图 6.1 所示。

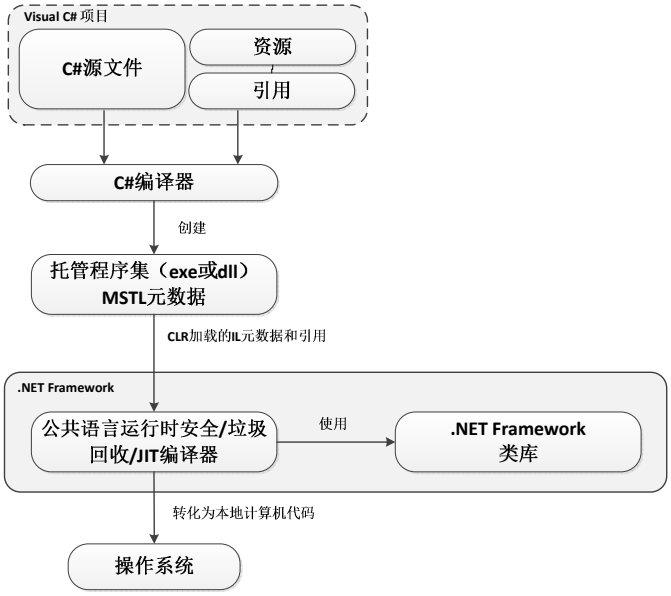

图 6.1　C#语言环境架构

6.1.2　.Net Framework

.NET Framework 是由微软公司发布的一种 Windows 平台下的软件开发框架，它允许 API 库和多种编程语言协同工作，从而创建基于 Windows 的应用程序。由.NET 构建的应用程序更易于部署和管理，也方便与其它网络系统集成，同时，.NET 运行时环境也对 Web Service 标准提供了支持，使得通过不同编程语言编写的应用程序之间可以

有效地交互数据。

.NET Framework 包含语言运行时环境和类库两部分。类库为程序员提供了一系列 API，这些 API 具有可重用的功能。类库中封装的 API 不仅能够被图形界面使用，还可以被命令行程序使用。另外，类库还为 ASP.NET 等 Web 应用程序提供了支持。语言运行时环境是.NET Framework 的基础，它主要提供版本兼容、内存管理、并行执行、多语言互操作以及重定向等服务。.NET Framework 将代码分为两类，分别是托管代码和非托管代码，其中，托管代码是以运行库为目标的，非托管代码是不以运行库为目标的。

.NET Framework 具有较高的安全特性。由于语言运行时环境提供了增强的安全规则，因此，在语言运行时环境上运行的托管代码必须首先经过安全验证，验证通过后才可以执行，而一些非类型安全的程序代码将不会执行。这就能够很好地避免常见的安全漏洞，如任意内存地址读写和缓冲区溢出等。虽然语言运行时环境也可以支持非托管代码的运行，但非托管代码不具有.NET Framework 的安全特性。

.NET Framework 的框架如图 6.2 所示。

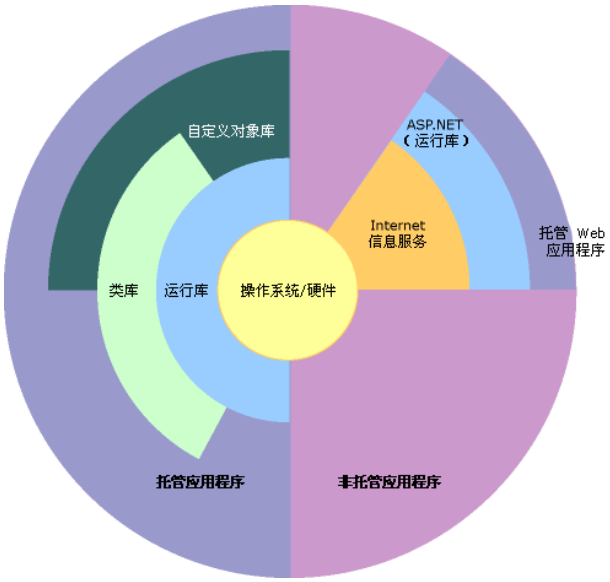

图 6.2　.NET Framework 框架

6.1.3　Visual Studio集成开发环境

Visual Studio 是当前最为流行的 Windows 平台下的应用程序集成开发环境，它是由微软公司推出的，其最新版本是 Visual Studio 2013 预览版。本文提出的 PDF 恶意代码检测模型 PDFCheck 是通过 Visual Studio 2010 进行开发的，Visual Studio 2010 发布于 2010 年 4 月 12 日，其界面被重新组织和设计，从而更加明了简洁。Visual Studio 2010 与 NET Framework 4.0 配套使用，并且支持 Microsoft SQL Server、IBM DB2 和 Oracle 等数据库产品。

6.2　关键模块实现

PDF 恶意代码检测模型 PDFCheck 包括基于主动学习策略的 PDF 文本检测子系统和 PDF 漏洞检测子系统，现对其关键模块实现进行介绍。

6.2.1　基于主动学习策略的PDF文本检测子系统实现

1）PDF 文件解析模块实现

PDF 文件解析模块是对 PDF 文件进行文本检测和漏洞检测的基础。在解析 PDF 文件时，首先通过 startxref 对根对象所在的 xref 的位置进行查找，然后由 trailer 得到根对象的序号，此时就精确定位到了 Catalog 根节点。接下来按照页面组、页面、资源、内容等顺序对 PDF 文件依次解析，提取 PDF 文件的对象和结构信息。

在实现 PDF 文件解析模块的过程中，创建了多个代码文件，其名称和作用如表 6.1 所示。

表 6.1　PDF 文件解析模块类名称及作用描述

类名称	作用
BasicObjects.cs	定义了 PDF 文件的基本对象
DisplayIndirectObject.cs	显示间接对象信息
ExceptionReport.cs	异常处理类，当发生异常时，显示有意义的错误消息
InflateMethod.cs	解压缩字节数组
LZWDecode	解密 LZW 压缩的字符串
PdfDocument.cs	表示 PDF 文件
PdfIndirectObject.cs	封装 PDF 间接对象
PdfParser.cs	解析 PDF 文件和 PDF 字节数组
ProgramState.cs	将程序信息保存到 xml 文件中

解析 Dictionary 的关键代码如下所示：

```
private PdfBase ParseDictionary()
{
   List<PdfPair> ResultDict = new List<PdfPair>();   //创建空数组
   NextChar = ReadChar();   //读取“<<”后的第一个字符
   for(;;)
   {
      SkipWhiteSpace();
      if(NextChar == EOF) throw new ApplicationException("Invalid dictionary (end of contents)");
      if(NextChar == '>') break;
      if(NextChar != '/') throw new ApplicationException("Invalid dictionary (name entry must have /)");
```

```
      StringBuilder Name = new StringBuilder();
      Name.Append(NextChar);
      while((NextChar = ReadChar()) != EOF && !IsDelimiter(NextChar))
         Name.Append(NextChar);
      PdfBase Value = ParseNextItem();
      if(Value.IsEmpty) throw new ApplicationException("Invalid dictionary (end of contents)");
      PdfPair Pair = new PdfPair(Name.ToString(), Value);
      Int32 Index = ResultDict.BinarySearch(Pair);
      if(Index >= 0) throw new ApplicationException("Invalid dictionary (duplicate keys)");
      ResultDict.Insert(~Index, Pair);
   }
   NextChar = ReadChar();
   if(NextChar == EOF || NextChar != '>') throw new ApplicationException("Invalid dictionary
      (missing terminating >>");
   NextChar = ReadChar();
   return(new PdfDict(ResultDict.ToArray()));   //退出
}
```

2）Javascript 攻击检测模块实现

①Javascript 代码提取

为提取完整的 Javascript 源代码，我们需要对 Adobe Reader 本地 JS 引擎中的关键函数（如 JS_EvaluateScript、JS_NewStringCopy、JS_EmitTree、parser、JS_GetScript 等）进行 hook，其关键代码如下所示：

```
jsd_GetScriptHook(JSDContext* jsdc, JSD_ScriptHookProc* hook, void** callerdata)
{
   JSD_LOCK();
   if( hook )
      *hook = jsdc->scriptHook;
   if( callerdata )
      *callerdata = jsdc->scriptHookData;
   JSD_UNLOCK();
   return JS_TRUE;
}
```

②Javascript 恶意代码检测

为对堆喷射攻击进行检测，需计算 Javascript 字符串的熵值，其关键代码如下所示：

```
void entropy(float x[],float y[],int s)   //对离散型随机变量的信息熵予以计算
{
   int i;
   double H_X=0,H_Y=0;
   for(i=0;i<s;i++)
      H_X+=-x[i]*(log(x[i])/log(2));
   for(i=0;i<s;i++)
      H_Y+=-y[i]*(log(y[i])/log(2));
```

```
}
void joint_entropy(float (*p)[u],int s,int t)   //对离散型随机变量的联合熵予以计算
{
   int i,j;
   double H_XY=0;
   for(i=0;i<s;i++)
      for(j=0;j<t;j++)
      {
         if(*(p[i]+j)<1e-6)continue;
         H_XY+=-*(p[i]+j)*(log(*(p[i]+j))/log(2));
      }
}
```

3）基于元数据和文件结构的特征提取模块实现

基于元数据和文件结构的特征提取模块用于对 PDF 文件的文本统计特性进行提取，需统计的文本特性有 count_font、count_javascript、count_js、count_stream_diff、pos_box_max、image_totalpx、producer_len、count_obj、pdfid0_mismatch、pos_eof_max、len_stream_min 等 120 个。为计算 count_font、count_javascript 等特征值，需对关键字的出现频率进行计算，关键代码如下所示：

```
private Dictionary<string, int> GetWordTermOccurence(List<Paragraph> paragraphs){
   Dictionary<string, int> counts = new Dictionary<string, int>();
   foreach (var p in paragraphs){
      foreach (var s in p.Sentences){
         for (int i = 0; i < s.Words.Count; i++){
            Word w = s.Words[i];
            CountTerm(counts, w.Stem);
            if (i > 0) {
               Word tm1 = s.Words[i - 1];
               string term = tm1.Stem + " " + w.Stem;
               CountTerm(counts, term);
            }
            if (i > 1) {
               Word tm1 = s.Words[i - 1];
               Word tm2 = s.Words[i - 2];
               string term = tm2.Stem + " " + tm1.Stem + " " + w.Stem;
               CountTerm(counts, term);
            }
         }
      }
   }
   return counts;
}
```

4）基于主动学习策略的 SVM 分类模块实现

对基于主动学习策略的 SVM 分类模块的实现利用了 C#编程语言的 SVM 类库。分类主程序的实现流程如图 6.3 所示。

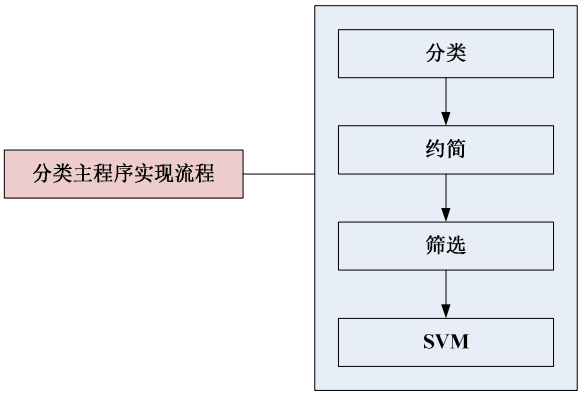

图 6.3　分类主程序实现流程

①分类：通过 Progress.classify()方法对测试样本和训练样本集合进行抽取，获得属性集合和属性值区间。

②约简：构造约简对象 Reduct，以 Reduct 的 calDependance()方法对属性依赖度 k 进行计算，返回依赖向量 dependance；以 Reduct 的 getDiffMatrix()方法对差别矩阵对象 DiffMatrix 进行构造；将专家参数初始化为 1。通过专家参数集合、依赖度 k 和差别矩阵 DiffMatrix 计算最终的权值向量 W。

③筛选：通过筛选对象 Filter 的 getSimilarity()方法，结合属性权值、值区间和测试样本的信息对每条训练样本的相似度 u 进行计算。

④SVM：根据计算得到的属性权值向量 W 和训练样本相似度 u 构造 SVM 分类器 Strategy。通过 Strategy.classify()方法将多类训练样本划分为单类样本集合 OneClass，然后使用 Strategy.predict()方法对测试样本进行预测。其中，将多类训练样本划分为单类样本集合的函数 Strategy.classify()的实现流程是：

a. 遍历训练样本；

b. 通过样本的类标签（决策属性值）判断该样本是否已经分类，如果是，将样本添加到已分类的单类样本集合中，否则构造新的样本集合 OneClass，将该样本添加进去。

Strategy.classify 的关键代码如下所示：

```
public void classify(List<DataRow> trainCollection){//按照类别对训练样本进行分类
   int sampleNum=ids.Count; //训练样本数
   double oldLabel = -1; //临时记录标签
   OneClass one = null;
   for (int i = 0; i < sampleNum; i++){
      DataRow sample = trainCollection[ids[i]]; //测试样本
```

```
        double label=double.Parse(sample[0].ToString()); //类标签
        if (label==oldLabel) {//同类
            one.addSample(sample, id_u[ids[i]]);
        }
        else { //新类
            oldLabel = label; //更新临时标签
            one = new OneClass(label); //构造类别
            one.addSample(sample, id_u[ids[i]]); //对样本及其相似度进行添加
            classes.Add(one);
        }
    }
}
```

对测试样本进行 SVM 预测的函数 Strategy.predict()的实现流程是：

a. 从两侧遍历排序后的单类样本集合；

b. 通过 OneClass.compare()方法将两个单类样本组合为二类 SVM 进行预测，对预测结果不是该类样本的标签的指针进行移动，直到结束；

c. 最后剩下的单类集合即为样本的预测结果。

6.2.2　PDF漏洞检测子系统实现

1）PDF 已知漏洞检测模块实现

为实现PDF已知漏洞检测功能，我们需要将与PDF漏洞相关的数据结构（包括BMP、IFF、TIFF、PCX、PICT、Universal3D、JPEG和TrueTypeFont等）的类型、名称、偏移量、值和长度等属性准确提取出来，这主要是由PDF文件解析模块完成的。

PDF已知漏洞检测模块采用了基于特征匹配的静态检测方法，首先将PDF漏洞检测规则库中的检测规则信息实现为C#源程序，然后对样本进行检测。如果PDF样本中有一个或多个数据结构的值命中了规则库中的一条或多条检测规则，那么就判定该样本文件存在PDF已知漏洞。PDF已知漏洞检测模块可以对19个已知PDF漏洞进行检测。

CVE-2011-2105是一个PDF缓冲区溢出漏洞，其检测规则如表5-1所示，该漏洞的逻辑较为复杂，下面对检测CVE-2011-2105漏洞的关键代码进行描述：

```
namespace GUT.DataFormats.PDF2{
  public partial class PDFDetectionLogic : PDF2{
    [CanDetect("CVE_2011_2436")]
    public void DetectCVE_2011_2436(){
      List<TBHDChunk> TBHDChunks =
        this.FindSubDataStructuresOfType<TBHDChunk>(true);
      List<RGBAChunk> RGBAChunks =
        this.FindSubDataStructuresOfType<RGBAChunk>(true);
      if (TBHDChunks.Count != 1 || RGBAChunks.Count != 1)
        return;
```

```
        foreach (TBHDChunk TBHDChunk in TBHDChunks){
          foreach (RGBAChunk RGBAChunk in RGBAChunks){
            if (RGBAChunk.X2.Value >= TBHDChunk.Width.Value){
              RGBAChunk.AddParsingNote(ParsingNoteRegular.PossiblyMalicious, "RGBA
              chunk specifies width larger than TBHD chunk", "CVE_2011_2436");
            return;
            }
          }
        }
      }
    }
}
```

2）PDF 漏洞利用关键代码检测模块实现

本节对堆栈检测和函数调用检测两种 ROP 链检测方法的关键代码进行描述，其中函数调用检测关键代码用于判断上一条指令是否为 call。

①内存检测

```
DWORD    StackBottom, StackTop;
GetStackInfo(&StackBottom, &StackTop);
if(((DWORD_PTR)pRSP < StackBottom) || ((DWORD_PTR)pRSP >= StackTop))
      Report(....);
```

②函数调用检测

```
BOOL ROP_CallerCheck(LPVOID ReturnAddress, LPVOID CriticalFuncAddress){
   for(int i=0;i<5;i++){
      //对返回地址上一条指令的地址进行计算
      LPVOID RreviousInsAddress = (LPVOID)((DWORD)ReturnAddress + g_CallInsLen[i]);
      int InsLen;
      //对返回地址上一条指令的类型进行判断，在变量ins_len中保存指令的长度
     if(CALL_INS == DecodeOpcode(PreviousInsAddress,&InsLen)){
        if(NULL == CriticalFuncAddress){   //如果上一条指令是call指令
           return TRUE;
        }
        else{ //如果上一条指令不是call指令
          if(CriticalFuncAddress == GetTargetBranchAddress(PreviousInsAddress)){
            return TRUE;
          }
        }
      }
   }
   return FALSE;
}
```

6.2.3　PDFCheck运行结果展示

下面对本文提出的 PDF 恶意代码检测模型 PDFCheck 进行测试：

1）模型主界面

PDFCheck 模型主界面如图 6.4 所示，它包括基于主动学习策略的 PDF 文本检测和 PDF 漏洞检测两项基本功能。

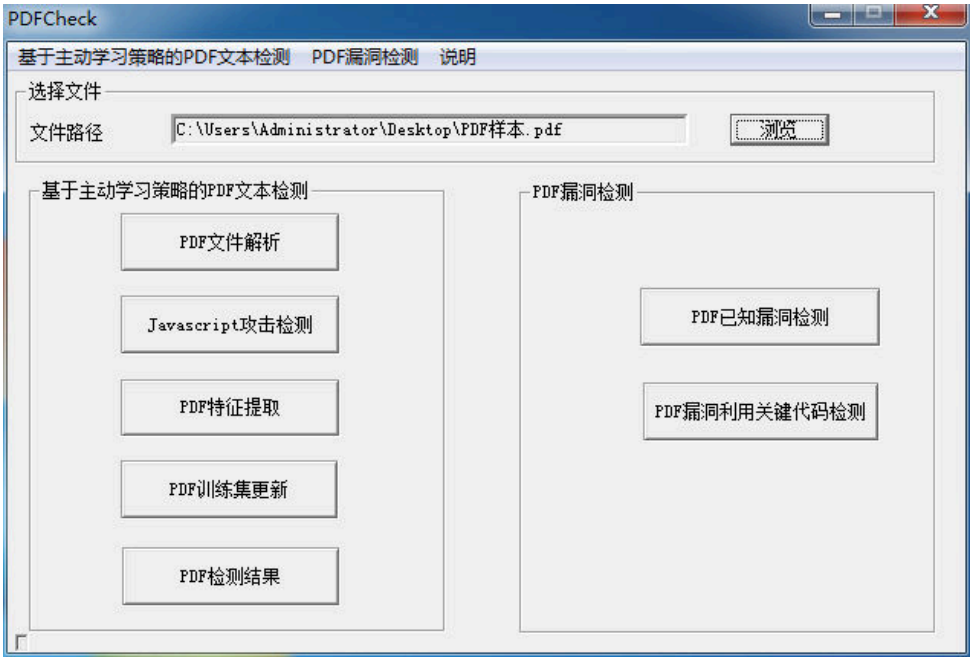

图 6.4　PDFCheck 主界面

2）PDF 文件解析

点击主界面上的“PDF 文件解析“按钮，运行结果如图 6.5 所示。模型以二维表格的形式将 PDF 文件的结构类型、结构名称、值、偏移量和长度等属性显示出来，并支持对属性的升序和降序排列。

321	Dictionary	/Resour...					249,894 (0x3D026)		
322	Dictionary	/Page				PageObj_322....	224,892 (0x36E7C)		
323	Stream	/Contents				StreamObj_32...	221,378 (0x360C2)	221,441 (0x36101)	3,433 (0xD69)
324	Dictionary	/Names					249,526 (0x3CEB6)		
325	Dictionary	/Names					249,587 (0x3CEF3)		
326	Dictionary	/Names					249,648 (0x3CF30)		
327	Dictionary	/Names					249,708 (0x3CF6C)		
328	Dictionary	/Names					249,770 (0x3CFAA)		
329	Dictionary	/Names					249,832 (0x3CFE8)		
330	Dictionary	/PTEX.I...					228,803 (0x37DC3)		
331	Dictionary	/ExtGSt...					229,068 (0x37ECC)		
332	Dictionary	/Font	/Type1				229,115 (0x37EFB)		
333	Dictionary	/FontDe...					229,544 (0x380A8)		
334	Stream	/FontFile3	/Type1C			StreamObj_33...	229,934 (0x3822E)	230,008 (0x38278)	2,995 (0xBB3)
335	Dictionary	/PTEX.I...					236,474 (0x39BBA)		
336	Dictionary	/ExtGSt...					236,739 (0x39CC3)		
337	Dictionary	/Font	/Type1				236,786 (0x39CF2)		
338	Dictionary	/FontDe...					237,215 (0x39E9F)		

图 6.5　PDF 文件解析运行结果

3）Javascript 攻击检测

点击主界面上的“Javascript 攻击检测”按钮，弹出新的对话框，如图 6.6 所示。Javascript 攻击检测模块可以查看 PDF 文件中嵌入的 Javascript 源代码和 Javascript 源代码在引擎中运行后产生的操作码，并可以利用计算字符串熵值的方法对堆喷射攻击进行检查。

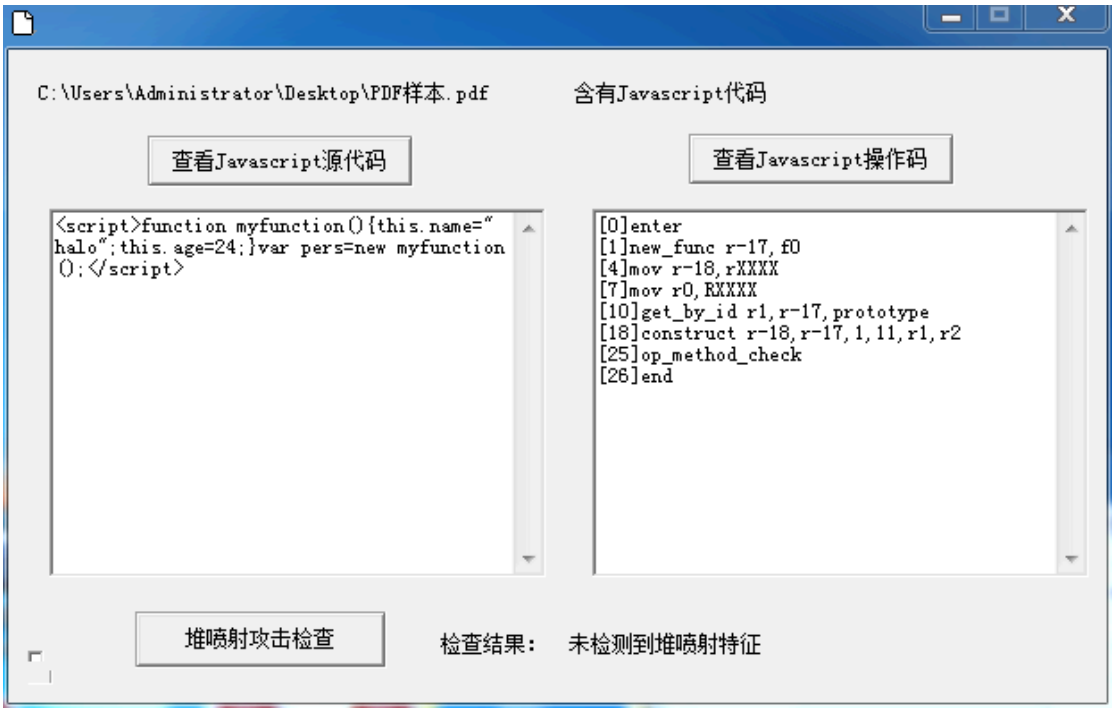

图 6.6　Javascript 攻击检测运行结果

4）PDF 特征提取

从 PDF 文件中提取的特征名称及特征值如图 6.7 所示。

特征名称	特征值
count_font	7
count_javascript	1
count_js	8
count_stream_diff	1
pos_box_max	5
image_totalpx	3
producer_len	3
count_obj	6
pos_eof_max	4
len_stream_min	721
count_endobj	1
ratio_imagepx_size	37
title_len	405
count_filter_obs	19
count_stream	1
producer_oth	12
createdata_tz	6

图 6.7　PDF 特征提取运行结果

5）PDF 训练集更新

更新 PDF 文件训练集的界面如图 6.8 所示，该界面允许用户添加正常训练文件和恶意训练文件。点击“SVM 分类检测”按钮后会输出检测结果，提示被检测文件是正常文件或恶意文件。

图 6.8　PDF 训练集更新运行结果

6）PDF 已知漏洞检测

使用 PDFCheck 对存在漏洞的样本文件“repro.pdf”进行测试，测试结果如图 6.9 所示。图中显示了漏洞编号和漏洞说明等信息。

解析类型	偏移位置	长度	漏洞类型	说明
Comment	0			Found unknown type -1.7
Error	0	7860		An exception was thrown ...
DefinitelyMalicious	730	6918	CVE_2011_2105	Found CID font with invalid...
Error	0	0		ParsingIncomplete: Trying ...
Error	7862			The prefix of a DataItem w...

图 6.9　PDF 已知漏洞检测运行结果

7）PDF 漏洞利用关键代码检测

对包含 ROP 链的 PDF 样本文件“roptest.pdf”进行测试，测试结果如图 6.10 所示。

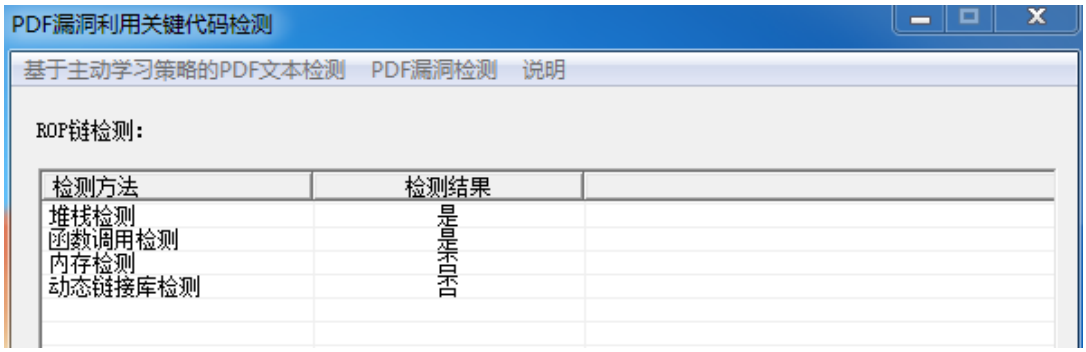

图 6.10　PDF 漏洞利用关键代码检测结果

6.3 模型比较分析

基于本文 3.5 节描述的测试数据和实验环境，将本文提出的 PDFCheck 与现有的 PDF 恶意代码检测模型 PJScan、ShellOS、MDScan、MPScan、PDFMS 和 PDF Scrutinizer 进行比较，其结果如图 6.11 所示。

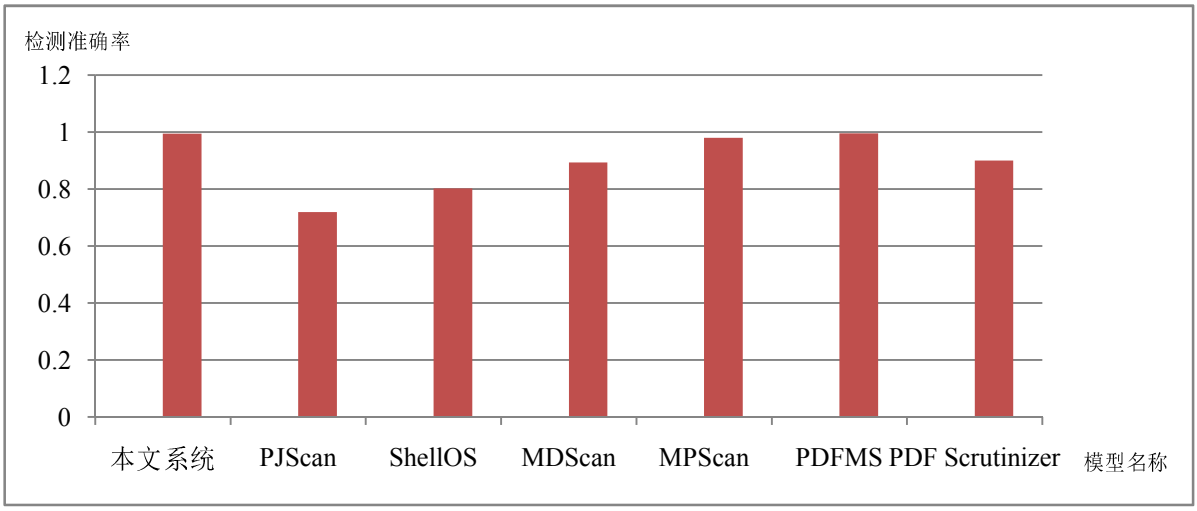

图 6.11 模型检测准确率比较

七种模型对单个文件的平均检测时间如表 6.1 所示。

表 6.1 模型检测时间比较

模型名称	单个文件平均检测时间（单位：ms）
本文系统	1750
PJScan	23
ShellOS	460
MDScan	1900
MPScan	1717
PDFMS	246
PDF Scrutinizer	2100

针对表 3.1 提出的恶意 Javascript 代码、代码混淆、反向模拟、远程代码加载以及远程 URI 解析等攻击技术，各个模型的检测能力如表 6.2 所示。

表 6.2　模型对特定攻击的检测能力比较

	PJScan	ShellOS	MDScan	MPScan	PDFMS	PDF Scrunizer	本文系统
恶意 Javascript	√	√	√	√	√	√	√
代码混淆		√	√	√		√	√
反向模拟	√					√	√
远程代码加载		√					√
恶意 URI 解析							√

由上述分析检测结果可知，本文提出的 PDF 恶意代码检测模型 PDFCheck 对恶意 PDF 文件的检测准确率约为 99.6%，明显高于 PJScan、ShellOS 和 PDFScrutinizer 等检测模型。在对单个文件的平均检测时间上，PDFCheck 高于 PJScan、PDFMS 等静态检测模型，与 PDF Scrutinizer、MDScan 和 MPScan 大体相当。表 6-2 描述了各个模型对特定攻击的检测能力，由于 PDFCheck 的 Javascript 攻击检测模块采用了 Hook Adobe Reader 本地引擎的代码提取技术和基于操作码特征匹配的恶意代码检测技术，因此，PDFCheck 能够识别这五种特定攻击方式，相比于其它模型，优势突出。

综上所述，本文提出的模型 PDFCheck 具有检测准确率高、检测方法多样、检测能力强等特点，并在时间开销和检测效果之间取得平衡。

6.4　PDF 漏洞检测能力测试

在本文 3.5.2 节描述的实验环境下，从可信数据源 VirusTotal repository、Contagio Project 和 Internet and Ben-Gurion University 中选择 200 个存在安全漏洞的 PDF 样本，将 PDF 恶意代码检测模型 PDFCheck 的已知漏洞检测模块与安全检测工具赛门铁克、BitDefender 进行比较，比较结果如表 6.3 所示。

表 6.3　PDF 漏洞检测结果比较

工具或模型名称	检出个数/总数	检出比率
PDFCheck	174/200	87%
BitDefender	156/200	78%
赛门铁克	139/200	69.5%

由检测结果可知，本文提出的 PDF 恶意代码检测模型 PDFCheck 对已知漏洞的检测能力明显高于安全检测工具 BitDefender 和赛门铁克。

6.5 本章小结

首先介绍 PDFCheck 的开发环境，然后对基于主动学习策略的 PDF 文本检测子系统和 PDF 漏洞检测子系统的关键模块分别予以实现，展示了模型的运行结果，并从检测精确度、检测特定攻击能力等方面将本文提出的模型 PDFCheck 与现有其它模型进行比较，比较 PDFCheck 和安全检测工具赛门铁克、BitDefender 的已知漏洞检测能力。

第七章　结论与展望

近年来，针对商业组织和政府机构的网络攻击事件层出不穷，APT 攻击时有发生。而 PDF 文件是 APT 攻击的主要载体，它通过执行嵌入在文件内部的恶意代码完成攻击过程。本文针对 PDF 恶意代码检测技术主要开展了以下研究工作：

1）从产生时间、检测方法和检测能力三个方面对现有的 PDF 恶意代码检测模型进行评价，分析它们存在的不足，以此为基础，提出一种新的检测模型 PDFCheck，该模型将主动学习策略、文本检测、漏洞检测等予以结合。

2）针对 Javascript 静态提取方法的不足，分析通过 Hook Adobe Reader 本地 JS 引擎的方式获取 Javascript 源代码、操作码和字符串的方法，并通过计算字符串的熵值，对堆喷射攻击进行检测。

3）采用基于元数据和文件结构的特征提取方法对 PDF 文件特征进行提取，测试特征数量改变时分类错误率的变化情况，从而选择最佳特征数量。

4）对 SVM 分类算法进行介绍，通过实验确定最佳参数 *C*；提出一种基于主动学习策略的 SVM 分类算法，描述算法流程和查询函数原理，对预测样本的分布规律进行测试，比较基于主动学习的 SVM 分类算法和基于监督学习的 SVM 分类算法，对实验结果进行分析。

5）结合 PDF 漏洞分析实例，对漏洞检测规则库进行构建，提出一种基于规则匹配的 PDF 已知漏洞检测方法，对测试结果进行分析；并基于 ROP 技术的原理，对堆栈检测、函数调用检测、内存检测和动态链接库检测四种 ROP 链检测方法进行描述。

6）使用 Visual Studio 2010 集成开发环境对 PDFCheck 的关键模块予以实现，将 PDFCheck 与现有的 PDF 恶意代码检测模型 PJScan、ShellOS、MDScan、MPScan、PDFMS 和 PDF Scrutinizer 进行比较，分析实验结果。

后续的研究方向主要有以下三个方面：

1）在 APT 攻击实施过程中，PDF 文件通常是以电子邮件附件的形式存在的，因此，可以考虑对邮件头部和邮件内容的特征进行提取，从而检测邮件的安全性。Miyamoto 等人提出了一种检测恶意邮件的方法，为我们接下来的研究提供了方向。

2）攻击者有可能设计出新的 PDF 文件攻击技术，这就需要不断地对本文提出的 PDFCheck 模型进行改进，以适应检测要求的变化。

3）现有的漏洞检测技术自动化程度不高，漏洞分析和检测规则构建等环节都需要人工参与，这就降低了漏洞检测的时效性。如何在理论上以形式化的方法对漏洞原理进行分析并构建检测规则，是我们的另一个改进方向。

参考文献

[1] Nick Sato. 91% of organisations hit by cyberattacks in 2013[EB/OL]. http://www.humanipo.com/news/37983/91-of- organisations-hit-by-cyber attacks-in-2013/, 2013-12-10.

[2] Andy O'Donnell. Tools and Utilities Commonly Used to Hack Computer Systems[EB/OL]. http://netsecurity. about.com/cs/ hackertools/a/aa030504.htm.

[3] Paloalto NETWORKS. What is an intrusion detection system ids[EB/OL].https://www.paloaltonetworks.com/resources/learning-center/what-is-an–intrusion-detection-system-ids.html.

[4] http://www.symantec.com/connect/articles/socialengineering-fundamentals-part-i-hacker-tactics.

[5] Infosecurity. 91% of APT attacks start with a spear-phishing email[EB/OL]. http://www. Infosecurity-magazine.com/view/29562/91-ofapt-attacks-start-with-a-spearphishing-email/,2012-11-28.

[6] http://www.f-secure.com/weblog/archives/00001676.html.

[7] Vatamanu C, Gavrilut, D, Benchea R. A practical approach on clustering malicious PDF documents [J]. Journal in Computer Virology.2012,8(4):151-163.

[8] Nir Nissima, Aviad Cohena, Chanan Glezerb, Yuval Elovicia. Detection of malicious PDF files and directions for enhancements: A state-of-the art survey[J]. Computers&Security.2015(48):246-266.

[9] Laskov P. Static detection of malicious JavaScript-bearing PDF documents[C]. Annual Computer Security applications conference. Pages 373–384, 2011.

[10] Srndic N, Laskov P. Detection of malicious pdf files based on hierarchical document structure[C]. Proceedings of the Network & Distributed System Security Symposium Ndss. 2013.

[11] Smutz C, Stavrou A. Malicious PDF detection using metadata and structural features[C]. Proceedings of the 28th Annual Computer Security Applications Conference. 2012.

[12] Maiorca D, Giacinto G, Corona I. A pattern recognition system for malicious pdf files detection. In: Machine learning and data mining in pattern recognition anonymous[M]. Springer-Verlag Berlin Heidelberg 2012:510-524.

[13] http://sourceforge.net/p/pjscan/home/Home/.

[14] Pareek H, Eswari P, Babu NSC, Bangalore C. Entropy and n-gram Analysis of Malicious PDF Documents[J]. International Journal of Engineering, 2013.

[15] Tzermias Z, Sykiotakis G, Polychronakis M, Markatos EP. Combining static and dynamic analysis for the detection of malicious documents[J]. Proceedings of Workshop on European Workshop on System Security Eurosec, 2011.

[16] Schmitt F, Gassen J, Gerhards-Padilla E. PDF Scrutinizer: Detecting JavaScript-based attacks in PDF documents[C]. Tenth International Conference on Privacy. IEEE Computer Society, 2012:104-111.

[17] Lu X, Zhuge J, Wang R, et al. De-obfuscation and Detection of Malicious PDF Files with High Accuracy[C]. Hawaii International Conference on System Sciences. IEEE Computer Society, 2013:4890-4899.

[18] Snow KZ, Krishnan S, Monrose F, Provos N. SHELLOS: enabling fast detection and forensic analysis of code injection attacks[J]. USENIX Security Symposium. 2011.

[19] Adobe PDF 101-Quick overview of PDF file format[EB/OL]. http://partners.adobe.com/public/developer/tips/topic_tip31.html.

[20] 丁晓煌.恶意 PDF 文档的静态检测技术研究[D]. 成都:电子科技大学, 2014.

[21] 周培和.PDF 文件格式漏洞挖掘系统的研究及实现[D]. 成都:电子科技大学, 2012.

[22] Margaret Rouse. sandbox[EB/OL]. http://searchsecurity.techtarget.com/definition/sandbox.

[23] Heap Spraying[EB/OL].http://baike.baidu.com/link?url=i6WK-kJ9n7pXiy9v3pXKN7tvd1ZlkUknO4dlnIIUJS7C0DRpbLN3Ocz-ijYwfi4d_NYHHJHsAxlNWu2wsd1CKK.

[24] Stevens D. Malicious PDF Documents Explained[J]. Security & Privacy IEEE, 2011, 9(1):80 - 82.

[25] Maiorca D, Corona I, Giacinto G. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious PDF files detection[C]. Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. 2013.

[26] Hamon V. Malicious URI resolving in PDF documents[J]. Journal of Computer Virology & Hacking Techniques, 2013, 9(2):65-76.

[27] [逆向工程]CVE-2013-0640 AdobeReader 任意代码执行漏洞分析[EB/OL]. http://forum.cnsec.org/thread-93167-1-1.html, 2014-5-3.

[28] 古河. CVE-2013-0640 漏洞利用分析.http://www.2cto.com/Article/201302/190540.html, 2013-2-22.

[29] Jnanamurthy H K, Warty C, Singh S. Threat Analysis and malicious user detection in reputation systems using Mean Bisector Analysis and Cosine Similarity (MBACS)[J]. IEEE India Conference, 2013:1-6.

[30] Shirley Liu X. An algorithm for finding protein–DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments[J]. Nature Biotechnology, 2002, 20(8):835-839

[31] http://www.baidu.com/link?url=J7556WfUNv7wO5liwxXqfy-XLSui0kkUdFhylqDtaBa

[32] 孟美华,王宏伟. PDF 文件文本内容提取的设计与实现[EB/OL].北京：中国科技论文在线[2009-10-15]. http://www.paper.edu.cn/releasepaper/content/200910-216.

[33] SpiderMonkey[EB/OL]. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey.

[34] 曾亮.嵌入式浏览器 Javascript 引擎的分析与优化[D]. 成都:电子科技大学, 2011.

[35] CVE-2010-3654 漏洞初步分析[EB/OL]. http://www.bkjia.com/xtaq/501297.html, 2013-11-29.

[36] Borg K. Real time detection and analysis of PDF-files. 2013.

[37] Schreck T, Berger S, Göbel J. Detection of Intrusions and Malware, and Vulnerability Assessment [M]. Springer Berlin Heidelberg, 2013.

[38] Wang X, Yu W, Champion A, et al. Detecting worms via mining dynamic program execution[C]. International Conference on Security and Privacy in Communications Networks and the Workshops. IEEE, 2007:412 - 421.

[39] Chen Z, Roussopoulos M, Liang Z, et al. Malware characteristics and threats on the internet ecosystem[J]. Journal of Systems & Software, 2012, 85(7):1650–1672.

[40] Chang C, Lin C. LIBSVM: a Library for Support Vector Machines[J]. Acm Transactions on Intelligent Systems & Technology, 2001, 2(3):389-396.

[41] Osuna E, Freund R, Girosi F. Training support vector machines: an application to face detection[C].

IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE, 1997:130 - 136.

[42] 施光莹.基于 Active SVM 算法的恶意网页检测技术研究[D]. 南京:南京理工大学, 2014.

[43] Adobe Acrobat and Reader CVE-2011-2437 Remote Heap Buffer Overflow Vulnerability[EB/OL]. http://ww w.securityfocus.com/bid/49579.2011-09-13.

[44] 王清.0day: 软件漏洞分析技术[M].电子工业出版社.2008:313-362.

[45] Detours[EB/OL]. http://research.microsoft.com/en-us/projects/detours/.

在学期间发表的学术论文

[1] **MengZheng**,LinYing,KangYan,YuQian.A parallel Programming Pattern based on Direct Acyclic Graph[J].Applied Mechanics and Materials.2012.12.（EI 检索）

[2] 孟正,郭荣华,文伟平.基于行为分析的木马攻击检测系统研究与实现[C].VARA 会议论文集.2014.9.

[3] 孟正,曾天宁,马洋洋,文伟平.Adobe Flash Player 漏洞简介与原理分析[J].信息网络安全.2014,(9):31-37.

[4] 孟正,梅瑞,张涛,文伟平.Linux 下基于 SVM 分类器的 WebShell 检测方法研究[J].信息网络安全.2014,(5):5-9.

[5] 林英,孟正,康雁,于倩.多核下一种线程调度算法的研究与实现[J].计算机技术研究与发展.2013.10,Vol.23,No.10,19-26.

[6] 梅瑞,孟正,霍玮.典型文档类 CVE 漏洞检测工具的研究与实现[J].信息网络安全.2014,(6):18-22.

[7] 姚洪波,孟正,文伟平.基于 P2P 的远程控制系统设计与实现[J].信息网络安全.2013.12

[8] 张涛,牛伟颖,孟正,梅瑞.基于 Windows 内核模式下进程监控的用户权限控制系统设计与实现[J].信息网络安全.2014,(4):13-19.

[9] 文伟平,郭荣华,孟正,柏晶.信息安全风险评估关键技术研究与实现[J].信息网络安全.2015,(2):7-14

[10] 郝增帅,郭荣华,文伟平,孟正. 基于特征分析和行为监控的未知木马检测系统研究与实现[J].信息网络安全,2015,(2):57-65.

致谢

首先，对我的导师文伟平副教授表示衷心地感谢，本论文是在文老师的悉心帮助和精心指导下完成的，从论文选题、初稿撰写到论文定稿，文老师提出了大量宝贵的建议，使我受益匪浅。他在论文的技术指导和组织结构方面为我提供了莫大的帮助。文老师以其严谨求实的治学态度、实事求是的科研作风、正直诚恳的待人风范和积极乐观的生活态度为我留下了深深的印象，将对我今后的工作和生活产生有益的影响。

接下来感谢我的团队——北京大学软件安全研究小组，感谢团队的梅瑞、宁戈、关通、张涛、马洋洋、张阳、张修、孟辉、柏晶、胡殿坤、谌力、张弛等同学，在整个研究生学习生涯中，他们给予我很多热情无私的指导和帮助。他们不怕艰难的意志，严谨的作风和积极的态度时时激励着我。在我遇到挫折时，他们一如既往地对我支持和帮助，他们是我学习、科研的坚强后盾。

此外，还要感谢北京大学软件与微电子学院的领导和老师们，研究生期间，他们从做人到研究学问再到生活的方方方面都给了我不少启迪，使我领悟到了很多受益终生的道理。

谢谢我挚爱的父母，是他们的支持成就了我今天的一切。在此向他们表达满满的祝福和最衷心的感谢。

衷心感谢评审本论文的各位老师、各位专家，感谢你们于百忙中抽出时间审阅本文。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　　月　　日