# 基于机器学习的PDF文件检测与分类器逃逸

## 摘要

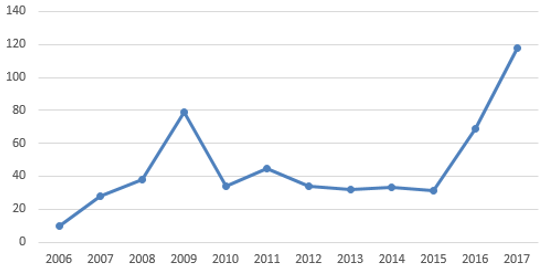
如今信息发展非常的迅速，基于恶意文档（如pdf）的检测越来越重要，文档类型的攻击具有针对性，并且更容易攻击成功。原因之一在于结构与形式的多样，更灵活地隐藏恶意有效载荷（例如，加密，隐藏为图像，字体或Flash内容）并逃避检测，攻击者手段之丰富（如通过电子邮件的附件传播，钓鱼网站加密混淆等 ）。其次网络攻击者正在转向基于文件的恶意软件，PDF规格已经改变。增加的脚本功能可以使文档以与可执行文件几乎相同的方式工作，包括连接到Internet的能力，运行进程以及与其他文件/程序进行交互。  
此研究项目的目标旨在为工业界和学术界提供一个基于AI的文档（PDF，word等）分类器，我们收集了200 000个样本进行检测，这些样本大部分是通过VirusShare 下载下来的，一部分来自于contagio，一部分通过爬虫收集了百度和搜狗的正常文本，类型多样和丰富，通过特征提取的特征计算，可以使其检测率达到99.81%，误报率0.01%，当个文件检测时间为毫秒级别。通过对文档领域特征的挖掘，并结合数据科学中的复杂建模，模型成功被构建。此模型可广泛应用于终端安全产品，又或者在邮件服务器，均是非常有意义的应用场景，AI文档分类器不仅限于pdf，如可扩展到word，execl，ppt等有结构的文档

## 简介

随着时间的推移，PDF规格和样式都在改变，变得丰富。增加的脚本功能可以使文档与可执行文件几乎相同的方式工作，包括连接到Internet的能力，运行进程以及与其他文件/程序进行交互。内容复杂性的增长为攻击者提供了更多的武器来发动强大的攻击，并且更灵活地隐藏恶意有效载荷（例如，加密，隐藏为图像，字体或Flash内容）并逃避检测。由于企业和个人用户系统未采用最新的浏览器插件（例如，Adobe的 Flash Player 和 Acrobat Reader）修补程序进行更新，这些攻击得以取得成功。

在2012年，开创性的漏洞攻击工具Blackhole专门针对Java，PDF和Flash文件。2013年，通过网络传输的不可执行恶意软件主要是针对Adobe Reader和Microsoft Office应用程序的PDF和Flash文件。在2014年发现的24个0day中，有16个是针对Adobe Reader和Flash Player的）如图1 是近几年来Adobe Reader在CVE上公布的一些漏洞统计，由此可以看出，从2015年开始关于Adobe Reader发现的漏洞就在线性增长，就表明了正对与PDF文档的攻击开始增多。

图1 Adobe Reader 在CVE中的增长趋势



在2017年的Symantec 的年度互联网安全威胁报告[13] （Internet Security Threat Report，简称ISTR）中显示中国依然是垃圾邮件的重灾区，中国在Malicious activity by source 总排名中名列前茅，其中在Spam zombies的统计中更是远超其他国家，排在了最前面的位置。

2016年，Symantec观察到大量通过恶意Office宏（W97M.Downloader和变体）和JavaScript下载程序文件（JS.Downloader和变体）分发勒索软件和网上银行威胁的电子邮件活动。 在他们之间，他们在2016年终端上的检测数量达到了700万，并且主宰了网络犯罪威胁的形势，特别是在当年下半年

随着Adobe Acrobat Reader（AAR）的普及，使其成为攻击者的首要目标，其次就是浏览器和操作系统内核，在引入了类似Chrome的沙盒机制（Sandbox）后，在Pwn2Own（全球最高规格的黑客大赛）竞赛中，单个漏洞的价值可高达6万美元。收集的恶意软件样本显示，许多Adobe组件已被利用，包括元素解析器和解码器，字体管理器]以及JavaScript引擎等。

Todo 近几年的文献与我们的工作2013-2018的

In This paper ,我们通过对不可执行的文档PDF的恶意行为进行研究，我们提供了一个案例研究的结果，我们在一个真正的基于学习的系统上实现了PDFrate，一个用于检测PDF恶意软件的在线服务[14]。对于任何提交的PDF文件，PDFrate 提供了对其恶意的概率估计。我们的研究以PDFrate为基本的分类结果为依据，使用一些模仿攻击，对PDFrate进行逃逸，但依然保留有恶意的属性。

我们在这项工作中使用的是一种名为Mimicus的PDFrate的开源重新实现，由Nedim Srndic和Pavel Laskov实施，模仿PDFrate进行恶意软件逃避实验[29]。 Mimicus接受了135个记录的PDFrate特征和与PDFrate.1相同的训练集的训练。Mimicus已被证明具有与PDFrate [29]几乎相同的分类性能。

我们已经分析并实现了PDFrate 的作者提出和评估的防御策略， 并却部署到邮件服务器上用于检测邮件附件。文献[29]的攻击方案假定攻击者利用一小部分信息特征。结果表明，通过将一小部分预期的攻击数据包含到训练集中，可以有效地阻止这种攻击。我们重建了这一攻击，并验证了原有防御战略的有效性。然而，这种主动防御最终只能有效抵抗攻击的“紧张”。每当执行的攻击与预期攻击不符时，主动防御的效果基本消失，检测精度降到10％以下

本文的主要贡献如下：

* 目前数据集从业界通用的万级别 到 达到 二十万 级别以上，其中恶意样本173036个，正常样本28332个。
* 选取了一组有用的和综合的静态特性（133个）以进行检测，使其可以有效的区分PDF 恶意与良性的样本
* 训练模型准确度高达99.81%，误报率0.16%
* 使用逃逸的方法生成7000个恶意文件成功逃逸工业界认可的PDFrate
* 成功实现了在几种情况下进行的回避攻击，并重新训练出一个新的模型（有一个3.1版本支撑）

## 相关工作(2018/05/22)

如今检测PDF文件的方法有很多，主要的方法可以分为两类，动态检测和静态检测，纵然这有一定程度上的重合。动态方法需要把文档放在某一个特定探针环境（instrumented environment）中打开运行；而静态方法则无需打开运行仅通过静态特征即可完成检测。

### 2.1.动态分析

一些核心想法促成了动态分析方法的兴起。最开始的工作是基于模拟的方法，此方法把可疑包（suspicious payload）使用抽象包执行（abstract payload execution）[36]或使用软件模拟[1，26]。然而，软件模拟并不能有效覆盖全部指令集。正因如此，有的文件就会逃逸检测。为了解决上述问题和增强可扩展性，最近提出的系统SHELLOS在可控shellcode执行的过程中[34]中使用硬件虚拟化技术以取代模拟方法。在一些常用的操作系统核心上，SHELLOS能有效地检测出shellcode（此shellcode在被程序任意分配的缓存之中）。然而，这么好的效果有一定的代价。尽管SHELLOS在检测网络层攻击时显示出优秀的吞吐量，但是检测恶意文档的时却遭受严重的延迟（在秒级别）。这种延迟造成的原因是检测位于内存缓存（memory buffers）中，此类检测需要程序首先开辟内存缓存，才能进行检测。

另外一种动态分析的方法聚焦于检测在运行JavaScript时的恶意行为。JSAND使用10个仔细提取的启发性特征去训练良性的JavaScript模型，然后使用此模型去检测与此模型基线有较大偏移的攻击[9]。一个相似的方法也被成功运用于ActionScript3恶意软件的检测当中[24]。CUJO建立在专门针对JavaScript的沙箱上，且可自动学习事件序列模型（models of sequences of events），这些模型均对JavaScript的解析器产生影响[31]。专门的Java Script动态分析方法与shellcode检测方法相比，有很大的性能提升（单个文件的检测时间控制在数百毫秒内），同时也保持了高准确度和低误报率。

### 静态分析

PJScan是第一个，成功实现基于JavaScript内容的PDF恶意软件检测[19]。为了提高效率，PJScan的JS提取器只搜索那些PDF标准预先指定好的位置。不幸的是，这种提取方法会在以下情况下有局限，攻击者可以将JS代码放在其他位置，通过PDF JavaScript API访问，并使用eval()函数调用来获取，以致来逃逸检测。

另外Maiorca等人提出的系统MALWARE SLAYER[23]，基于模式识别。这里的模式是指使用PDFID工具从PDF文档中提取的文本关键字。它在真实PDF数据集上展现出高准确度和低误报率。这种方法被PDFID自身的函数所局限，且不能处理多版本（multiple revision numbers）和在对象流（object stream）中的隐藏对象（object hidden）。

PDFrate是一个相对成熟基于机器学习的静态PDF分类器。此分类器使用简单的PDF附带信息（meta info）和字节层面（byte-level）的文件格式。此PDFrate在大数据集中有出色的分类表现[33]。然而，此分类器不能提取对象流（object streams），在检测时这个特征就不能加入到训练样本中。

Maiorca等人。也研究了对PDFrate和其他PDF文档分类器的逃逸[26]，[27]。他们提出 反向模仿技术。为了使内容看似良性（如Mimicus所做的那样），他们将恶意内容嵌入到良性PDF中，而不是将内容添加到恶意文档中，而是尽可能少地修改。反向模仿攻击实施针对PDFrate的独立逃避方法, Maiorca等人提出了三种不同的逃避方案。在EXEembed方案中，恶意可执行文件被植入到现有的良性PDF文档中。打开文档时会执行恶意软件。这些文档利用CVE-2010-1240。在PDFembed方案中，恶意PDF被嵌入到良性PDF中。这些嵌入式文档在文档打开时自动呈现。为了评估，Maiorca等人 将利用CVE-2009-0927的文档嵌入到现有的良性PDF文档中。最后，在JSinject场景中，恶意JavaScript（与PDFembed嵌入式文档中使用的相同）直接注入根良性文档。

### 动静结合

另外两个须提及的贡献是把静态和动态分析技术结合起来。MDScan[37]对PDF文件进行静态分析，目标是提取所有的JS代码簇（chunk），这些代码簇可以作为执行的入口（entry point）。直到现今为止，一个特定用途的解析器（parser）被MDScan开发，这个解析器被用于抓取在文件里面的其他信息，这些信息包括被交叉引用图表（cross-reference table）所忽略的对象以及潜在的恶意对象。被抓取的代码会在JavaScript引擎中被执行（此引擎会模拟Acrobat Reader引擎）。在控制执行中（controlled execution），所有的内存缓存都会被一个工具所检测，此工具基于二进制模拟（binary emulation NEMU）以用于shellcode检测。

在ZOZZLE[10]一文中，静态与动态部件的角色发生了变化。ZOZZLE的动态部件会在运行前提取JavaScript的片段，这些JavaScript源于IE浏览器中的JavaScript引擎，此引擎能自然而然地解决JavaSript代码混淆（obfuscation）的问题。ZOZZLE的静态分析部分使用Bayesian分类器，此分类器建立在对检测到的JavaScript源代码进行混合分析（syntactic analysis）的基础上。

### 2.4.机器学习

传统的恶意PDF 检测方法有基于病毒检测、基于签名的的检测方法[14]等，这些方法存在识别率不高、无法及时更新恶意代码等问题。机器学习技术为恶意PDF 检测提供了新方向，PDF 文档的检测研究大多采用PDF 文档内容或结构为特征，利用随机森林、SVM、决策树等分类器构建PDF 检测器,。例如，Charles 等人提出通过随机森林检测含恶意代码的PDF 文件技术PDFrate，作者从PDF 文档元数据以及文档结构中提取了135 个特征，使用已标记特征的训练数据，并采用10 倍的交叉验证，生成具有多个分类树的分类器，从待测PDF 文档提取特征，评估森林中的每个树，最后投票决定其分类。该方法初始训练过程计算开销较大，但一旦分类器构建完成，对待测的PDF 文档的分类速度很高，我们同时也对这135个特征进行了提取，利用随机森林的算法对文档进行预测分类。

[14]Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0R: Detection of Malicious PDF-embedded JavaScript Code through Discriminant Analysis of API References. In Proceedings of the Artificial Intelligent and Security Workshop (AISec), 2014.

对相关工作的比较表明，在当前用于文档恶意软件检测的静态和动态系统间有明显的权衡（trade-off）。纵然动态分析系统表现出高准确度和低误报率，这种优势是以高延迟、明显的性能瓶颈和需要特定的受控制系统（instrumented environments）为代价。在本文中提出的新方法，旨在从静态方法填补空隙。此技术通过从静态方法开始，增强检测性能同时保持系统设计的简约和计算的高效。为了达到这样的目标，我们使用了一个现成的解析器，构筑关于“Comprehensive Static Analysis of PDF”的方法论。更进一步地，我们对样本潜在的逃逸策略有特别的关注，且用实验去评估所提及的方法对特定攻击策略的健壮性。

## PDF文件分析

作为一种结构化的文件格式，一个PDF文档是由一些称为“对象”的模块组成的。每个对象都有数字标号，这样的话可以这些对象就可以被其他的对象所引用。这些对象不需要按照顺序出现在PDF文档里面，出现的顺序可以是任意的，比如一个PDF文件有3页，第3页可以出现在第1页以前，对象按照顺序出现唯一的好处就是能够增加文件的可读性，对象的信息以偏移+索引的形式保存在交叉引用表内

### 3.1 PDF文件结构分析

可移植文档格式（PDF）是一个开放标准，发布为ISO 32000-1：2008 [ [9](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR9) ]。PDF文件格式包含以下4个部分：

**文件头——指明了该文件所遵从的PDF规范的版本号，它出现在PDF文件的第一行。**

**文件体——又称对象集合，PDF文件的主要部分，由一系列对象组成。**

**交叉引用表——对对象进行随机存取而设立的一个间接对象的地址索引表。（实际以偏移+索引的方式储存对象地址，下文会提及）**

**文件尾——声明了交叉引用表的地址，即指明了文件体的根对象（Catalog），从而能够找到PDF文件中各个对象体的位置，达到随机访问。另外还保存了PDF文件的加密等安全信息。**

**其中的一些对象包括以下几个**

* /xref 交叉引用表，描述每个间接对象的编号、版本和绝对的文件位置。PDF文档中的第一个索引必须从版本65535的0号对象开始，标识符/xref后面的第一个数字是第一个间接对象（即0号对象）的编号，第二个数字是/xref（交叉引用表）的大小。
* /Page指明PDF文件的页数，大多数恶意PDF文件仅仅只有一页
* /Encrypt指明PDF文件有数字水印或者是被加密过的。
* /ObjStm是object streams的数量。object streams是一个可以包含其他Object对象的数据流对象。
* /JS与/JavaScript指明PDF文件中含嵌有JavaScript代码。通常恶意的PDF文件都嵌套有JavaScript代码，这里一般都是利用JavaScript的解析漏洞或者使用JavaScript来实现堆喷射（heap spray），也有很多正常的PDF文件里会含有JavaScript代码
* /AA、/OpenAction和/AcroForm指明当查看PDF文件或者PDF的某页时会有动作随其执行，几乎所有嵌有JavaScript代码的恶意PDF文件都有自动执行JavaScript代码的动作(action)。如果一个PDF文件包含有/AA或/OpenAction自动执行动作的关键字段，而且含有JavaScript代码，那么这个PDF文件就极有可能是恶意的PDF文件
* /URI 如果你要在PDF文件中执行打开网页的动作就需要这个关键字段
* /Filter 一般为FlateDecode则是使用了Zlib压缩解压缩算法。
* /JBIG2Decode指明PDF文件使用了JBIG2压缩。虽然JBIG2压缩本身可能会有漏洞（CVE-2010-1297）。但/JBIG2Decode关键字段并不能说明PDF文件是否可疑
* /RichMedia Flash文件
* /Launch执行动作(action)数量

图2 PDF文件格式图示



文件尾说明了根对象的对象号，并且说明交叉引用表的位置，通过对交叉引用表的查询可以找到目录对象(Catalog)。这个目录对象是该PDF文档的根对象，包含PDF文档的大纲(outline)和页面组对象（pages）引用。大纲对象是指PDF文件的书签树；页面组对象（pages）包含该文件的页面数，各个页面对象(page)的对象号。因此，PDF阅读器通常从文件末尾开始阅读PDF以提高效率

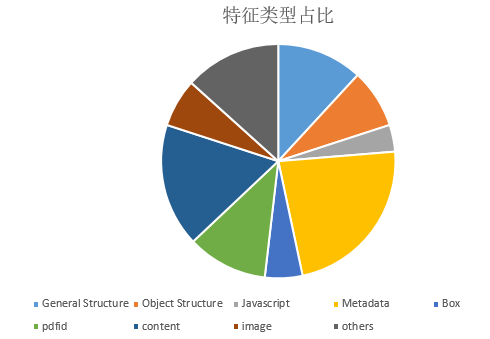
页面（page）对象为PDF中最重要的对象，包含如何显示该页面的信息，例如使用的字体，包含的内容（文字，图片等），页面的大小。里面的信息可以直接给出，当然里面的子项更多的是对其他对象的引用，真正的信息存放在其他对象里面。页面中包含的信息是包含在一个称为流（stream）的对象里，这个流的长度（字节数）必须直接给出或指向另外一个对象（包含一个整数值，表明这个流的长度）。可见stream流对象我们恶代分析需要获取的重点。

### 3.2 PDF特征选择

据调查显示，根据PDF结构路径在文件中存在最能表明文件是良性的，或者缺少文件结构等表明文件是恶意的。例如，恶意文件不太可能包含元数据以最小化文件大小，打开文档时不会跳转到文档中的页面，并且格式不正确，因此它们缺少路径，例如/ Type和/ Pages　／count

特征识别的理念是产生尽可能多的特征,并确在用于训练分类的时候对分类效果有一定的帮助，以尽可能地对文档的元数据和结构进行提取并矢量化，这些特征反映metadata的属性，例如每个字段中的字符数; objects/streams，比如每个对象的大小和数量; boxes和images，例如每个的大小和位置; 数据编码方法，如使用每种数据编码方法; 和对象类型，例如加密对象的计数。 总共选择了133功能。

图3 特征分布图



在特征提取的时候提及特征提取的方法有基于结构和Metadata的，也有基于结果和内容的，还有Javascript的，这里我们可以看到，提取结构是非常重要的，我们就结合以上的三种方法是发现可以是检出率高达99%，并且提取结构的主要优势是在于1）。提取方式已经较为成熟，使用工具就可以很快的将文件结构提取出来，2）.提取文件结构的时间也会相比其他的提取方式要快，需要更少的时间就可以将文件的特征提取出来，3）.以现有的技术提取出文件结构出来，已经可以达到一个很好的精确度了，经过多次测试都在99%以上。

我们通过计算良性样本和恶意样本的每一个特征的平均值，发现正常样本的均值会在一定的区间内浮动，而恶意样本的特征相对于正常样本来说就会有很大的差距，如表1所示：

count\_font,count\_box 等这些特征，在正常样本中就会有很多关于font ,box这些结果，毕竟文档类型文件的主要的就是使用文字和图片来描述信息，可是恶意文档一般就可以不用那么多的文字信息，而是直接将恶意代码嵌入到文档当中，就可以达到目的

经过我们的统计，在统计同一个页面的obj对象时，良性样本和恶意样本也会有一定的差距，一般来说良性样本的obj对象要比恶意样本的要多很多，可是我们发现，如果obj在同一个页面会突然增多，那么这个文件就会显示出极为可疑

其中count\_endobj 与count\_endstream是经常会用与结构混淆的特征，按照正常的PDF文件来说，一个对象开始就会就会有一个endobj，可是有的PDF恶意文件为了故意混淆分类器，就会近可能少的使用endobj和endstream,这样解析器在解析的时候就不能完整的获取到整个对象，或者解析失败，以至于直接逃过分类器。这个是恶意文件嘴长使用的一个逃逸方法，在后来的解析中也会提及如果防止这些文件逃逸的策略。

另一个就是count\_js，这个是显而易见的一类特征，对 于恶意文件主要就是嵌入一些JS代码来执行一些恶意行为，或调用系统命令，一个恶意文件JS的代码就会比良性样本的数量要多一些，其中还有一部分用来混淆和加密的JS的大小与良性样本也存在一定的差异

 还有一个重要的差异是count\_acroform\_obs 的统计数量也会有所不同，AcroForm是PDF Specification 1.2 中引入的这种表单是从用户那里通过交互方式收集信息的字段集。这类表单包括[数据表示](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E8%A1%A8%E7%A4%BA" \t "_blank)、数据捕捉和数据编辑，它还可以进行动态交互：动态功能从具有动态计算、验证及其他特性的交互式、可编辑的表单，到服务器生成的、机器填充的表单等各不相同。同时动态布局：表单可以自动重新调整自身以适应用户或[外部数据源](https://baike.baidu.com/item/%E5%A4%96%E9%83%A8%E6%95%B0%E6%8D%AE%E6%BA%90)（例如[数据库服务器](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E6%9C%8D%E5%8A%A1%E5%99%A8)）提供的数据。例如，如果从服务器中检索到的数据有 100 行，则表单显示 100 行。并且复杂：此功能包括单页静态表单，[动态文档](https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E6%96%87%E6%A1%A3)将基于数据内容进行装配，而大型生产运行包含数十万个事务。基于以上几个特点，就很容易成为攻击者混淆加密的地方，所以恶意文件在计算AcroForm值得时候，会比正常样本要多一倍多。

Table 1: Mean Values between Benign & Malware Samples

|  |  |  |
| --- | --- | --- |
| Features | Benign Files | Malware Files |
| **count\_font** | 14.646365 | 0.551510 |
| **count\_acroform\_obs** | **700** | **1400** |
| **count\_box\_a4** | **12001** | **200** |
| **count\_box\_legal** | **395040** | **0** |
| count\_box\_letter | 7291529 | 866773 |
| count\_box\_other | 32.184318 | 1.749550 |
| count\_box\_overlap | 1000 | 0 |
| **count\_endobj** | **95.800680** | **9.685137** |
| **count\_endstream** | **30.436044** | **3.785357** |
| **count\_page\_obs** | **8001** | **16003** |
| count\_image\_large | 110711 | 400 |
| count\_image\_med | 465247 | 6401 |
| **count\_image\_small** | **915892** | **12002** |
| count\_image\_total | 36.56366 | 0.30006 |
| count\_image\_xlarge | 300 | 0 |
| count\_image\_xsmall | 21.64216 | 0.11202 |
| **count\_js** | **0.714071** | **1.012402** |
| **count\_obj** | 100.969997 | 12.014803 |
| count\_objstm | 1.572057 | 0.153831 |

## 数据集与数据清理

### 数据集

PDF文件被分类为良性或恶意两类：benign PDF和Malicious PDF。

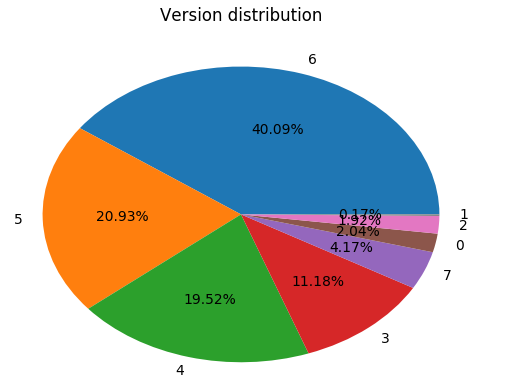
目前训练和测试样本一共20万，如表8 所示：

表8 训练样本的版本分布情况

|  |  |  |
| --- | --- | --- |
| Malicious | Benign | total |
| 173036 | 283323 | 201368 |

我们的数据集主要是从VirusShare下载下来的156035个。大小有6.8G，还有9000个正常样本来自于contagio,其他还有2万的平衡样本来自mimicus [15] ，包括有contagio的5,000个良性文件和5,000个恶意文件，还有5000的google的良性样本，和VirusStotal的5000个恶意样本。其他的数据集是在搜狗和百度上通过爬虫爬取下来的。其中mimiucus中的数据集中的两个用于[23]中的PDF RATE 实验性评估：Contagio ，可供下载。

表3 训练样本的版本分布情况



### 数据筛选

对所有收集起来的数据进行初步筛选,我们最开始使用的解析器是peepdf ，首先对VirusShare 的数据集进解析，可是有一半的文件不能通过peepdf 解析到，进而无法提取到特征，进入到机器学习的计算当中，我们的Model1 中的数据就只正对与可以解析的10万个文件进行训练与预测。

之后我们针对第一次未分析到PDF文件进行重新复检，发现有些恶意的PDF文件部分结构不全导致分类器无法识别，可是依然有恶意性质保留，无法正常打开，缺少某些对象或结束符。于是我们对所有的文件重新筛选，经过第一步的筛选之后 156035 个样本中只有83442个通过了此次筛选，也就是说还有72593个文件不能被解析器识别，所以我们就专门针对没有被解析到的这些数据进行了分析，幸运的是，Nedim ˇ Srndic 在研究PDF的时候也遇到了这个问题，他们通过修改PDF解析器的规则 ，可以成功的解析那部分不容易被解析的文件，并且读取了PDFrate[11] 中的135个特征，其余的202个特征中的67个并没有透露，之后关于PDF的解析与分析都会以这135个特征为基线来做其他的分析。

基于以上的数据清理，我们就选取了平衡数据集来重新训练模型Model2 ,其中的数据集包括2万的正常样本和2万的恶意样本（这些恶意样本是Model1 之前没有解析到的），我们训练的时候通过几个算法对比后，使用了随机森林算法对模型进行训练，直到模型的准确率达到99.99%后，我们就把数据集增加到十万，对模型重新训练，当达到十万级别的数据集后，模型的准确率达到了99.81%。

## 分类评估方法

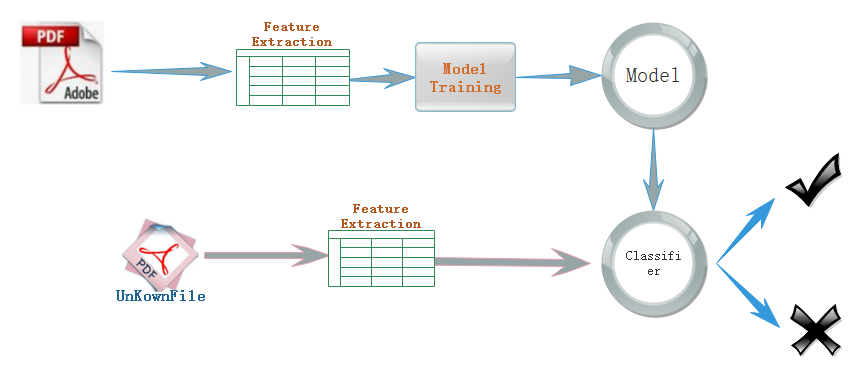
### 训练分类模型构建

所提出的用于基于结构的检测恶意PDF文档的方法包括以下两个步骤，如下图所示：

1.    提取结构特征。作为基本的预处理步骤，PDF文档的内容被解析并转换成特殊的形式，包 - 路径，以一种定义明确的方式表征文档结构。

2.   学习和分类。检测过程是由恶意和良性的PDF doc

图4 机器学习的基本框架



在这项工作中，我们关注分类器，这是一种特殊类型的学习系统，它将新数据分为两个或多个预定义的类别。分类器通常通过计算一些数字或概率分数并将其与固定阈值进行比较来做出预测。旨在操纵分类器的对手的目标是将其混淆 为提供错误的分类。对于二元分类问题，错误分类被称为false positives和错误否定false negatives。从对抗的角度来看，有关基于学习的系统的信息越多，系统成功的机会就越大。

构成每个基于学习的分类器系统的基本组件是：

•    分类器使用的一组特征

•    训练数据集用于分类器训练

•    分类算法及其参数。

因此，为了对手的利益，最大化他们对目标分类器组件的知识。例如，对手*一个*谁知道某个分类的功能集和训练数据集有逃避它比对手的机会较高*乙*谁只知道功能集。从这个意义上说，两个对手*A*和*B*在不同的*逃避场景下*运作。逃避场景是从对手的角度来看逃避的问题。它以结构化的方式描述对手可用的分类器系统信息：概述对手对于特征集合，训练数据集合和分类算法的误报率是低还是高。

### 吞吐量

在处理大量PDF数据时，高吞吐量是一个重要的考虑因素，就像V IRUS T OTAL 那样。我们的系统旨在处理这种负载，并利用现代计算机系统的并行处理能力。我们测量了使用随机森林，决策树和SVM 对十万的数据集进行特征提取，学习和分类所花费的时间如 表7 。训练预测是在一个4核4G的CPU的一个Ubuntu 的系统下进行的。

特征提取是最耗时的操作，因为它需要从硬盘加载所有的PDF文件。并对文件进行逐个解析，我们测试的数据有10万，一共5.8G。对这些数据进行了两次解析，一共耗时22分钟36秒。然后对文件进行训练和预测，如图7 是不同算法之间的训练时间和预测时间，以及精度的统计。由表我们可以看出随机森林和决策树的预测时间都快于SVM，但是决策树的训练时间会比随机森林快很多，这个是由算法本身的因素决定的，并且随机森林的准确率是明显高于其他算法的。

SVM默认核函数 参考https://blog.csdn.net/batuwuhanpei/article/details/52354822

表7 训练时间与预测时间

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training | Classification | accuracy |
| Random Forest | 56s | 1s | 99% |
| Decision tree | 4s | 1s | 97% |
| SVM | 58m 18s | 12s | 75% |

### 算法选择

AI引擎的核心就是算法，我们选取了几个实用性较好的几个算法作为计算，这几个算法包括KNN邻近算法，NNET(神经网络)，RF（随机森林）SVM（支持向量机）经过多次的训练与分类实验，最后随机森林以其准确率较高，误报率极低，和低延时，并且技术时间快的优势，被用于我们最后模型的核心算法。

我们通过对四种算法的准确率和鲁棒性进行比较，发现随机森林的准确率和鲁棒性要好很多

表8 算法准确率对比

|  |  |  |  |
| --- | --- | --- | --- |
| KNN | NNET | RF | SVM |
| 0.971292 | 0.824163 | 0.996411 | 0.752392 |

我们使用了KNN邻近算法，NNET(神经网络)，RF（随机森林）SVM（支持向量机）四个算法来做训练，如表6 可以看出RF预测精度会比其他三个算法高，并且随着特征减少依然可以有很好的一个准确率。为了评估这个几个算法

#### 随机森林(Random Forest,简称RF)

对观察到的文件进行分类，首先对提取出来的特征作为训练数据集生成一个分类器，在这个时候，随机森林（random forests）就排上用场了，因为随机森林可以很有效的对文件进行分类，同时误报率极低，并却易于使用，对于一个初学的人来说，也可以很快的使用随机森林来对数据进行分类。随机森林分类方法给出的结果是基于很多颗分类树判断结果的集合展现，每一个决策树都是在训练数据中随机选择生成的，因此，随机森林总的来说是一个集成分类器，他使用 bagged training data, 通过选择随机选择的特征子集,，并使用该节点的训练数据,，确定每个节点处的最佳分割来创建树中的每个节点。此外，每棵树都是基于一个独立的特征子集，最后，在分类过程中每一个树的投票来确定最终结果。

PDFrate采用随机森林[14]，一种集合学习方法，包括独立训练的决策树的数量*t RF*。在训练步骤中，使用CART方法学习每棵树，但只使用可用训练样本的一个子集。通过从训练数据中随机抽样一定数量的树来为每棵树生成一个不同的子集，用替换 - 称为引导聚合或装袋的过程。当新的决策节点被添加到树中时，只考虑*f个RF*特征的随机选择的子集，其中*f RF*小于功能的总数。通过在给定的新数据点上的所有决策树中进行多数投票作出决定。随机森林以其卓越的泛化能力和对数据噪声的稳健性而闻名。PDF RATE 使用Leo Breiman's和Adele Cutler的原始随机森林实现的R端口，作为软件包RANDOM F OREST。*t RF*和*f RF*分别是称为*ntree*和*mtry*的RANDOM F OREST的 参数。值*ntree* = 1000 和*mtry* = 43由PDF RATE使用。

所有这三个分类的PDF部署率，即接受了有关的那些*Contagio*， *GMU*和*社区*数据集，生产他们的结果是他们的决策函数在区间输出，即真正的价值[0 *，* 1] 表示的百分比将提交的文件标记为恶意的决定。在[23]中没有给出阈值来确定一个文件应该被认为是恶意的。请注意，通过提供此百分比值而不是二元决策PDF RATE揭示了更多关于分类引擎的信息，而不是决策所必需的信息，从而使对手能够在制定逃避方法时作出更明智的决定。

输入为样本集D={(x,y1),(x2,y2),...(xm,ym)}D={(x,y1),(x2,y2),...(xm,ym)}，弱分类器迭代次数T。

　　　　输出为最终的强分类器f(x)f(x)

　　　　1）对于t=1,2...,T:

　　　　　　a)对训练集进行第t次随机采样，共采集m次，得到包含m个样本的采样集DmDm

　　　　　　b)用采样集DmDm训练第m个决策树模型Gm(x)Gm(x)，在训练决策树模型的节点的时候， 在节点上所有的样本特征中选择一部分样本特征， 在这些随机选择的部分样本特征中选择一个最优的特征来做决策树的左右子树划分

　　　　2) 如果是分类算法预测，则T个弱学习器投出最多票数的类别或者类别之一为最终类别。如果是回归算法，T个弱学习器得到的回归结果进行算术平均得到的值为最终的模型输出。

#### 决策树（Decision Trees）

决策树比其他算法优胜的地方在于她良好的可解析行，如下是在NOTEBOOK上用scala推算时生成的树，IPython notebook目前已经成为用Python做教学、计算、科研的一个重要工具。该工具既能动态驱动，还具备可重复生成的特点。我们在试验中也使用这个工具对我们提取的数据进行测试，使用数据集是十万级别，训练时间仅10.881s ,预测时间5.783s , 并且分类效果也达到了 99.49%，训练深度为5 ，共41个节点，如表三所示

如下图所示，表5 是分类器在对特征进行分类的过程中生成的一颗树的其中一个枝子，其中就是以Version\_Action\_JA 的这个特征作为结点的，这个特征是区分正常与恶意样本的一个关键特征，因为很多恶意文件的恶意代码都会嵌入到 Action\_JS 这个对象当中，而在正常样本很少会有使用这个对象。下一个节点就是Version\_Streams,统计版本中流的数目，恶意文件的流对象一般会比正常文件的流对象要多一些，当然这个只是作为一个枝子，在后面还要进行合并判断，比如版本号，Metadata 的长度，Javascript的数量与分析，等等，多个特征合并计算权值后投票对文件进行判断。

表5 决策树的分类过程图

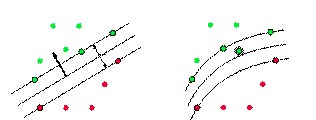


#### SVM（Support Vector Machines）

支持向量机（SVM）是另一种流行的机器学习算法[12]。它的主要几何思想，如图4所示，是将一个超平面拟合到数据中，这样两个类的例子就以最大可能的边界*M*被分开。

Vapnik等人在多年研究统计学习理论基础上对线性[分类器](https://baike.baidu.com/item/%E5%88%86%E7%B1%BB%E5%99%A8)提出了另一种设计最佳准则。其原理也从线性可分说起，然后扩展到线性不可分的情况。甚至扩展到使用非线性函数中去，这种分类器被称为支持向量机（Support Vector Machine，简称SVM）。

图5 SVM分类图示



SVM学习的有效实现可以在各种机器学习包中获得，包括WEKA和SHOGUN。在我们的实验中，也使用SVM参与测试。

### 分类结果

在我们的实验当中我们一共对模型进行了4次更新，其中最开始的模型（model1）是通过使用peepdf为解析工具，然后通过一些特征计算与特征量化，使其可以用于机器学习训练的数据，当我们的特征数提取到133个的时候，我们提取的特征包括有基于结构的（count\_font、size、count\_startxref），内容信息的（title\_oth、subject\_lc）和metadata(producer\_oth、producer\_len)的一些静态属性,此时准确率就达到了99.30%，可是经过我们的研究发现，因为在一开始我们使用peepdf进行解析的时候，并不所有文件都解析到参与到，所以在第二阶段

如下表是更新后模型的一个改进与与更新，在特征与数据集上有所改进，具体的就是Model 2 在特征选取上更为多，由之前手动提取的133个特征，增加到135个，并且也是在学术界中比较认可的135个特征集来训练。

Model2 是一个中间测试的过度模型，Model2.1训练数据集为十万级别，并且所有测试样本都是病毒的变种样本，模型检测率为99.81%

图9 各个模型参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| model | 特征数目 | 数据大小 | 检测率 | 误报率 |
| model 1 | 133 | 10w | 99.30% | 0.0054 |
| model 2 | 135个 | 4W | 99.99% | 0.00012 |
| model 2.1 | 135个 | 10w | 99.81% | 0.00086 |
| model 3.0 | 135个 | 21W | 99.20% | 0.00024 |

### 模型评估：

为了评估我们的检测模型的预测性能，我们将整个数据集随机分为两部分：训练（90％）和测试（10％）。 我们对每个组合重复该过程10次以上来验证结果。 这种方法有助于评估一个给定的检测方法在没有先验信息的情况下的鲁棒性。

我们对特征有效性进行研究，将所有特征进行一个重要性排序，然后依次将最重要的特征删减，后进行重新训练预测，测试结果如图9所示，我们可以观察到当特征减少了100个之后，模型检测率依然可以达到90%，说明模型具有很好的鲁棒性和抗攻击能力。

我们通过改变输出类别概率的阈值来生成ROC曲线。如表9和表10所示，我们的检测率超过99％，RF分类器的误判率低于0.01％。 如图10所示，ROC曲线下面积约为1，表明有效的特征和鲁棒性模型。 换句话说，功能的静态挖掘在区分良性和恶意文件方面非常有效。 我们可以使用静态分析有效识别未知文件的好坏，以发现可疑信息。

图9 特征自减后的识别率



ROC准确度

图10 ROC曲线图



## 逃逸检测

### 分类器逃逸分析

为了系统地探索针对分类器系统的逃避攻击，基于知识分子拥有的关于分类器系统三个组成部分的知识量，利用如下的逃避场景分类。

● 在情况F中，只有特征集可用于敌手，在不同程度上。攻击者可能会意识到某些或全部特征，错误地将过时的特征视为正在使用，能够读取子集或全部特征或者能够以不同程度修改某些或全部特征。然而，为了能够修改样本并进行规避，需要操纵足够的特征子集。

● 在FT情况下，除了已知的特征之外，攻击者还可以利用目标分类器训练数据集的知识。数据集可能完全或部分泄漏，从而在生成成功的攻击样本的过程中实现更准确的决策。

● 在场景FC中，攻击者知道特征集以及关于分类器的一些细节，例如类型，参数或具体实现。根本没有关于训练数据集的信息而没有替代数据集的对手几乎不知道分类器的优点。使用替代数据集，他们可以训练正确类型的替代分类器，但是这种近似的准确性取决于收集的数据的质量。这种攻击也可以离线执行，类似于基于替代分类器的其他攻击。

● FTC如果知道所有三个分类器组件的细节，那么对手就有最大的机会躲避目标分类器。在这种情况下，他可以在线下完全重现在线分类器，只有在找到足够好的规避样本时才提交攻击结果。离线模仿攻击或离线分类器特定攻击击败离线分类器也具有很强的击败在线分类器的可能性。

我们复现了这4种攻击场景，。他们的名字描述了对手可用的信息。如果对应于分类器组件特征集的字母F，T或C中的任何一个分别存在于场景的名称中，则训练数据集和分类器算法分别存在于对手所具有的给定分类器组件的知识水平这种情况很高，否则低。名为O的情景指的是敌手对所有三个分类器组件都知之甚少的情况。

说清楚FC和FTC的攻击量的一些不同，pdfrate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attack method | Test file | attaktMoel2.1 | FN | recall |
| F | 2157 | 71.18% | 27.11% | 71.18% |
| FC | 240 | 2.92% | 97.08% | 2.92% |
| FT | 4196 | 84.25% | 15.30% | 84.25% |
| FTC | 600 | 15.83% | 84.16% | 15.83% |

### 模型自我修复

攻击的一个故事

知道特征，攻击，然后修复

图12 前30个重要特征分布图



### 攻击结果

攻击样本选取：

攻击样本的选取只要针对Model逃逸的样本做进一步的筛选，有很大一部分样本，在结构上会有一些省略或对象不完整，导致解析器无法解析，就无法提取到完整的特征，因为这个原因分类器就无法对此样本进行近一步的分类判断，在很大程度上就逃逸了分类器，其他的的一些逃逸方法还会有编译器混淆，加密，多层包含，等等，

在Model1中最开始有接近一半的样本（7万多的样本逃逸分类器）没有被解析到，进而没有参与训练分类，可是样本依然有恶意的代码，保留有其恶意的属性，在Model2 训练的时候，就在这7万多的样本中选取逃逸较多的那些，多次测试后使用FP的样本，使用Nedim Srndic在SP会议中的基因编程技术，生成新的变种来测试分类器的鲁棒性。我们最后使用2038个逃逸样本来测试Model，然后再训练一个新的Model2.1, 如下图，很明显的，在经过编译后，Model的攻击相对来是成功的，几乎80左右的样本逃逸。

在本次攻击的的几个案例中，我们主要正对上面说到的4种情况进行病毒变种与分类器逃逸攻击。

### 预防措施

在我们上一个实验中，我们已经研究了[23]中提出的防御机制对我们的逃避技术的鲁棒性。为了设定基线，我们使用在*Contagio*数据集上训练过的随机森林分类器，完全按照Smutz和Stavrou提出的模拟攻击和防御技术（参见第IV-D节）重现模仿攻击和防御技术。我们的分类器按照最重要的顺序排列了以下30 个特征：

模仿攻击唯一可以想象的特征是具体特征的经验支持，即攻击改变了特定特征的文件的百分比。有几乎每次攻击都会改变的45个特征。剩下的23个特征很少被修改，很可能是由于相反的变化方向（就是说，35个特征只能在我们的设置中增加），或者由于所请求的变化的不可行性。

基于学习的方法不安全的根本问题在于特征的设计。机器学习在各种信息系统中越来越受欢迎 - 这远远超出了安全 - 主要是因为它能够预测，或多或少的成功，从*副作用的原因*。使机器成为这种概括性的能力

1 动态分析的检测：

我们使用了一些具有恶意链接调用的PDF进行动态检测，我们可以看到正常的恶意文件是很少有URL调用的，恶意的PDF文件一般会通过使用暗链连接外部数据，从而下载恶意软件，对客户端进行有目的性的攻击。在对这部分文件进行检测时，因为文件往往是有针对性的，所以对一些特定的攻击与攻击环境很难模拟到，以至于在沙箱里面运行的时候，检测不到恶意行为，于此同时以工业性的角度来看，动态检测不仅会占用大量资源，内存，这样一来就与沙箱检测的环境要求还是很高的，并且沙箱检测的时间远远的高于静态检测的时间，不利于在线检测的一个实时返回的一个结果。对于静态检测说，主要针对静态特征的提取，以及文件格式和内容的静态代码分析，

## 应用场景

嵌入AI PDF分类器的智能防火墙

按照模块化的思想，我们把基于AI的PDF分类器看作一个独立的检测模组，并把此检测模块集成到安全产品防火墙中。一个面世30多年的老式安全产品，如何为其插上AI安全的翅膀，蓝盾在这方面走在业界的前列。

在当今边界安全产品中，如防火墙，对于网络应用层（第七层）的恶意文件扫描已经是国际标准。业界对此功能的需求极为严苛，一个优秀的功能模组通常要求单个文件检测（扫描）时间（即延迟）维持在毫秒级别，对PDF文件的检测准确度需在99%以上，而误报率则不能大于0.01%。

低延迟的功能要求是因为文件扫描功能，在硬件层，是串联于整个文件检测的流水线中，高延迟会显著增加丢包率，造成严重数据丢失，这对于安全设备是不可容忍的。这些年，随着恶意软件的急速增加，先前基于规则匹配的引擎从 标配 开始力不从心：一方面为了达到产品需求的高准确度，需要大量的安全分析人员编写规则和旧设备的规则库；另一方面随着核心规则库的扩大，规则匹配算法的时间也呈现指数级别增长。这两方面都促使我们探索更新更好的AI引擎。

综上所述，我们把基于AI的PDF分类器串联集成到防火墙中，成功替代了原本基于规则的引擎。纵然上述两引擎均基于静态分析技术，AI引擎带来的优势却是非常明显的。一方面，此引擎本无需频繁更新，据实验数据，正常AI引擎的平均更新周期为半年，而规则引擎则为2周；第二方面是AI引擎卓越的低计算资源消耗，据实验数据，AI引擎能在预测时稳定于约1/3的CPU占用，约50%的内存占用。CPU计算力的消耗主要在特征提取和计算最终结果概率上，内存的消耗则来自AI模型自身在预测时需位于内存中。

在此真实防火墙应用场景中，我们根据AI模型不同的输出概率以及原因，分情况触发后续多样的动作。举例说，当输出概率大于0.9时，表明基于静态分析方法有较大把握预测此样本为正（即恶意），故会第一时间触发安全产品，如防火墙，对此所属链接进行封锁（block）操作；相反地，当概率小于0.1时，表明模型有较大把握预测此样本为负（即善意），安全产品默认允许此所属链接。

有趣的是当输出概率在灰色区间时如[0.1, 0.9]所触发的行为，我们会把样本上传到云中作动态分析，此分析包括沙箱，安全分析师和威胁情报等。静态分析的优势是速度快，但对于某些具有强逃逸能力（如混淆加密等）的样本会有一定的局限；动态分析的优势是准确度高，分析深入但速度慢，占用资源较高。

综上所述，我们可知，基于AI的静态分析和动态分析技术各有千秋，我们通过把两种技术合理运用，能较好地满足真实应用场景下的产品需求。

## 总结

在本论文中，我们详细介绍了基于AI的PDF恶意文档分类器的设计与实现，并在此基础上，结合具体真实应用场景，成功搭建了一个基于动静态分析的恶意文档立体检测平台。实验数据表明，在十数万级文档数据集中，我们能达到大于99%的准确度和小于0.01%的误报率。更进一步的，且在实际运行时，CPU和内存的 时空效能比（time & space performance）比旧有基于规则的模型，有显著提升。

本文除了使用大量数据描述 安全AI（apply AI tech. to security），把 AI安全(securing the model & the associate) 也放在了同等重要的位置上。我们通过大量的实验，模拟了（1）攻击者通过对恶意样本的增删改（如添加好的PDF对象），以混淆分类器，大量逃逸的目的；（2）分类器经自身修正，通过重新训练模型，去除已被攻击者所利用的特征（feature set been exploited），以维持模型的健壮性。

基于AI的文档分类器是社会工程学、病毒分析等领域的重要研究课题。在未来，我们还会尝试解决以下研究问题：

* 基于深度学习的恶意PDF文档检测
* 动静态分析引擎的调优
* 对于Microsoft Office等其他文件格式的支持（如docx，pptx等）

## 参考文献

1. Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovann Vigna. Efficient Detection of Split Personalities in Malware. In Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February–March 2010
2. Curtis Carmony, Mu Zhang, Xunchao Hu, Abhishek Vasisht Bhaskar, and Heng Yin. Extract Me If You Can: Abusing PDF Parsers in Malware Detectors. In Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2016
3. Nedim ˇ Srndic and Pavel Laskov. Detection of Malicious Pdf Files Based on Hierarchical Document Structure. In 20th Network and Distributed System Security Symposium (NDSS), 2013
4. Nedim Srndic and Pavel Laskov. Practical Evasion of a Learning- Based Classifier: A Case Study. In Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland), San Jose, CA, May 2014
5. Nedim ˇ Srndic and Pavel Laskov. Mimicus: A Library for Adversarial Classifier Evasion. <https://github.com/srndic/mimicus>.
6. Nedim Šrndic and Pavel Laskov . Hidost: a static machine-learning-based detector of malicious files, Šrndi′c and Laskov EURASIP Journal on Information Security (2016) 2016
7. Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2016.
8. VirusTotal. Free Online Virus, Malware and URL Scanner.https://www.virustotal.com/.
9. Stephan Chenette. Malicious Documents Archive for Signature Testing and Research - Contagio Malware Dump. http://contagiodump.blogspot.de/2010/08/ malicious-documents-archive-for.html.
10. D. Maiorca, D. Ariu, I. Corona, and G. Giacinto, “A Structural and Content-Based Approach for a Precise and Robust Detection of Malicious PDF Files”, in Proceedings of the 1st International Conference on Information Systems Security and Privacy. ScitePress Digital Library,2015, pp. 27-36.
11. Charles Smutz and Angelos Stavrou. Malicious PDF Detection Using Metadata and Structural Features. Technical report, 2012.
12. C. Cortes and V. Vapnik. Support vector networks. Machine Learning, 20:273–297, 1995.
13. <https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf>
14. Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0R: Detection of Malicious PDF-embedded JavaScript Code through Discriminant Analysis of API References. In Proceedings of the Artificial Intelligent and Security Workshop (AISec), 2014.PDFrate
15. https://github.com/srndic/mimicus

[27] Nedim ˇ Srndic and Pavel Laskov. Mimicus: A Library for Adversarial Classifier Evasion. https://github.com/srndic/mimicus.

[29] Nedim ˇ Srndic and Pavel Laskov. Practical Evasion of a Learning-Based Classifier: A Case Study. In 35th IEEE Symposium on Security and Privacy (Oakland), 2014.

## 附录：

|  |  |
| --- | --- |
| feat | imp |
| count\_font | 0.108772 |
| count\_javascript | 0.087662 |
| size | 0.079581 |
| count\_obj | 0.069801 |
| count\_endobj | 0.059954 |
| producer\_oth | 0.05054 |
| producer\_len | 0.04837 |
| pdfid1\_num | 0.039756 |
| producer\_dot | 0.037493 |
| count\_box\_other | 0.03686 |
| count\_stream | 0.036113 |
| count\_endstream | 0.0302 |
| count\_js | 0.020743 |
| pdfid0\_len | 0.019087 |
| producer\_lc | 0.01906 |
| producer\_mismatch | 0.018668 |
| len\_obj\_max | 0.018317 |
| len\_stream\_avg | 0.017994 |
| len\_obj\_avg | 0.016931 |
| pdfid0\_num | 0.016363 |
| producer\_uc | 0.016284 |
| pdfid1\_len | 0.015949 |
| len\_stream\_max | 0.013086 |
| producer\_num | 0.011488 |
| count\_startxref | 0.009243 |
| len\_obj\_min | 0.008178 |
| count\_page | 0.008106 |
| pdfid1\_mismatch | 0.007617 |
| pdfid0\_mismatch | 0.006044 |
| createdate\_version\_ratio | 0.005681 |
| creator\_len | 0.005603 |
| moddate\_version\_ratio | 0.004428 |
| title\_oth | 0.004373 |
| creator\_lc | 0.003601 |
| box\_other\_only | 0.003494 |
| creator\_uc | 0.003382 |
| pdfid\_mismatch | 0.00239 |
| moddate\_mismatch | 0.002291 |
| count\_box\_letter | 0.002237 |
| createdate\_mismatch | 0.001802 |
| moddate\_tz | 0.0017 |
| count\_eof | 0.001628 |
| subject\_lc | 0.001521 |
| title\_num | 0.001473 |
| len\_stream\_min | 0.001459 |
| title\_len | 0.001409 |
| count\_trailer | 0.001283 |
| pdfid1\_uc | 0.001269 |
| createdate\_tz | 0.001172 |
| title\_dot | 0.00116 |
| subject\_len | 0.001011 |
| title\_uc | 0.000923 |
| version | 0.000897 |
| title\_lc | 0.000852 |
| moddate\_ts | 0.000791 |
| box\_nonother\_types | 0.000789 |
| creator\_oth | 0.000768 |
| author\_len | 0.000736 |
| count\_xref | 0.000735 |
| subject\_oth | 0.000664 |
| count\_action | 0.000659 |
| createdate\_ts | 0.00063 |
| pdfid0\_uc | 0.000599 |
| delta\_ts | 0.000594 |
| count\_acroform | 0.000591 |
| author\_uc | 0.000591 |
| image\_totalpx | 0.000526 |
| creator\_dot | 0.000494 |
| delta\_tz | 0.000457 |
| count\_objstm | 0.000434 |
| creator\_num | 0.000433 |
| author\_lc | 0.000378 |
| author\_oth | 0.000319 |
| pdfid0\_oth | 0.000309 |
| count\_stream\_diff | 0.000293 |
| count\_image\_total | 0.000258 |
| creator\_mismatch | 0.000248 |
| author\_mismatch | 0.00024 |
| title\_mismatch | 0.00023 |
| keywords\_len | 0.000219 |
| pdfid1\_oth | 0.000209 |
| author\_num | 0.000203 |
| count\_image\_large | 0.000187 |
| keywords\_lc | 0.000175 |
| pdfid0\_lc | 0.000138 |
| count\_image\_small | 0.000114 |
| keywords\_oth | 0.000109 |
| author\_dot | 8.83E-05 |
| ratio\_imagepx\_size | 8.65E-05 |
| pdfid1\_lc | 6.91E-05 |
| subject\_mismatch | 6.74E-05 |
| subject\_uc | 5.21E-05 |
| keywords\_uc | 5.03E-05 |
| keywords\_mismatch | 4.02E-05 |
| count\_javascript\_obs | 2.06E-05 |
| count\_page\_obs | 1.51E-05 |
| count\_box\_a4 | 1.24E-05 |
| image\_mismatch | 1.02E-05 |
| subject\_dot | 9.83E-06 |
| count\_image\_med | 9.75E-06 |
| count\_font\_obs | 9.20E-06 |
| count\_action\_obs | 8.90E-06 |
| company\_mismatch | 7.60E-06 |
| keywords\_num | 7.54E-06 |
| count\_js\_obs | 6.23E-06 |
| subject\_num | 5.21E-06 |
| count\_image\_xsmall | 3.92E-06 |
| count\_acroform\_obs | 0 |
| count\_box\_legal | 0 |
| count\_box\_overlap | 0 |
| count\_image\_xlarge | 0 |
| count\_objstm\_obs | 0 |
| createdate\_dot | 0 |
| keywords\_dot | 0 |
| moddate\_dot | 0 |
| pdfid0\_dot | 0 |
| pdfid1\_dot | 0 |
| pos\_acroform\_avg | 0 |
| pos\_acroform\_max | 0 |
| pos\_acroform\_min | 0 |
| pos\_box\_avg | 0 |
| pos\_box\_max | 0 |
| pos\_box\_min | 0 |
| pos\_eof\_avg | 0 |
| pos\_eof\_max | 0 |
| pos\_eof\_min | 0 |
| pos\_image\_avg | 0 |
| pos\_image\_max | 0 |
| pos\_image\_min | 0 |
| pos\_page\_avg | 0 |
| pos\_page\_max | 0 |
| pos\_page\_min | 0 |
| ratio\_size\_obj | 0 |
| ratio\_size\_page | 0 |
| ratio\_size\_stream | 0 |