1. **相关工作**

如今检测PDF恶意文档的方法众多，主要的方法可分为两类，动态与静态检测。动态分析需要把文档放在某一个特定探针环境（instrumented environment）中打开运行；而静态分析则无需运行，仅通过静态特征分析即可。表1 基于之前的研究列出了对现有方法的总结。

表1 PDF分析技术对比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 方法 | 分析重点 | 检测技术 | 是否有解析器 | 是否是ML | 是否依赖 | 逃逸 / 缺陷 |
| 静态分析 | JavaScript | Lexical 分析 [27] | 是 | 是 | 是 | 代码混淆，加密 |
| JavaScript | Token 聚类 [59] | 是 | 是 | 是 |
| JavaScript | API 调用分类 [14] | 是 | 是 | 是 |
| JavaScript | Shellcode and opcode 签名 [31] | 否 | 否 | 是 |
| Metadata | Linearized object path [36] | 是 | 是 | 是 | Mimicry[53], Reverse mimicry[35] |
| Metadata | 分层结构检测 [33, 52] | 是 | 是 | 是 |
| Metadata | 基于内容和Metadata [46] | 是 | 是 | 是 |
| Content | 基于结构和内容解析2015[10] | 是 | 是 | 是 |
| 综合 | 结合上述几种技术解析分类 [34] | 是 | 是 | 是 |
| 动态分析 | JavaScript | Shellcode and opcode签名检测 [58] | 是 | 否 | 是 | 多元化的JS引擎分析 基于Non-script的攻击 |
| JavaScript | 已知的攻击模式 [45] | 是 | 否 | 是 |
| JavaScript | 内存访问模式 [48] | 是 | 否 | 是 |
| JavaScript | 常见 maldoc 行为分析 [29] | 否 | 否 | 是 | Zero-day 漏洞利用 ROP 和 JIT-Spraying |
| JavaScript | 独立平台的tap point标识技术[2] | 是 | 否 | 是 |
| 文档类型 | 异常内存访问约束变量 [62] | 否 | 否 | 否 |
|  | 平台多样性 | 系统平台多样性的漏洞利用[30] | 否 | 否 | 是 |  |

### 2.1 动态分析

动态分析最开始的工作基于模拟，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，以上方法并不能有效覆盖全部指令集。正因如此，有的文件会成功逃逸检测。为了解决上述问题和增强可扩展性，系统ShellOS在可控shellcode执行的过程中[34]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在被程序分配的缓存之中）。然而，好的效果有一定代价。尽管ShellOS在检测基于网络层的攻击时显示出优秀的吞吐量，但检测恶意文档时却有严重的延迟（延迟下降到秒级别）。这种延迟造成的原因是：检测位于内存缓冲（memory buffers）中，需要程序首先开辟内存缓冲，才能进行检测。

另外一种动态分析方法聚焦于检测在 JavaScript运行时的恶意行为。JSAND使用10个有效特征去训练良性JavaScript模型，然后使用此模型去检测与其基线有较大偏移的攻击[9]。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）[31]。专门的JavaScript动态分析方法与Shellcode检测方法相比，性能提升大，单个文件的检测时间可控制在百毫秒级别内）。

### 静态分析

PJScan是首个成功实现基于JavaScript内容的PDF恶意文件分类器[19]。为了提高效率，PJScan的JS提取器只搜索那些PDF标准有预先指定的位置。其缺点是显而易见的，攻击者可将JS代码放在其他非PDF标准指定位置，再通过API访问，并使用eval()函数调用来获取，以此成功逃逸检测。

Maiorca等人提出的MALWARE SLAYER系统[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。此系统在真实PDF数据集上展现出高准确度和低误报率。但这种方法同时也被PDFID自身的函数所局限：它不能处理多版本（multiple revision numbers）和在对象流（object stream）中的隐藏对象（hidden object）。

PDFrate是一个相对成熟的，基于机器学习的静态PDF分类器。此分类器使用PDF附带信息（meta info）和字节层面（byte-level）的文件格式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 stream），导致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]并提出反向模仿技术。为了使内容看似良性，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，此攻击利用CVE-2010-1240。

### 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。被抓取的代码会在JavaScript引擎中被执行。在控制执行（controlled execution）中，所有的内存缓冲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。

### 机器学习（把此部分嵌入到静态分析中）

Charles等人[14]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。

### 对抗性学习

在许伟林等人2016发表的关于自动逃逸分类器的一文中[7],他们针对两种PDF恶意文档分类器（**PDFrate和Hidost**）生成变异样本用于逃逸。这样一来即使分类器的检测率为99%，也可以通过这个方法，多次变异后成功逃逸分类器。对抗性学习是一个非常值得关注的问题，同时也是衡量模型鲁棒性的关键。Maiorca等人[34]关注JavaScript和元数据，并将许多上述启发式算法融合到一个程序中，以提高逃避弹性。相反，反向模仿[35]攻击尝试将恶意内容嵌入到良性PDF中，注意尽可能少地修改恶意内容。Mimicus [53]实现了模仿攻击并通过向文档添加空的结构和元数据项而对现有的maldoc进行更好的显示，使其看起来更像良性，而对渲染没有实际影响。

由于Adobe Reader的不断开发，恶意文件（maldoc）检测已经成为一个紧迫的问题。尽管已经提出了许多解决方案，但最近的作品突出了一些常见的缺陷，就如上面所提到的几种攻击方法，针对这一点Meng Xu等研究者就提出了一个检测PDF恶意文件的新视角：平台的多样性，通过操作系统设计和恶意样本与良性样本运行时的行为差异，从系统语义到堆对象元数据结构，进一步展示如何阻止攻击者发现错误，避免其逃逸。