1介绍

闯入计算机系统最有效的工具之一仍然是恶意软件，即恶意软件。自从个人计算曙光以来，它一直是一个众所周知的瘟疫，但恶意软件在近十年来已经发展出几种阴险的特性来满足犯罪业务的需求。其中之一是用于在企业和个人之间交换文件的众所周知格式的文件感染。这种感染为攻击者提供了以下好处：

1. 1。

诱使用户打开文档比启动可执行程序更容易。

1. 2。

近年来，文档查看器中出现了一系列新的漏洞，这是因为文档格式的复杂性导致了其高度复杂性。

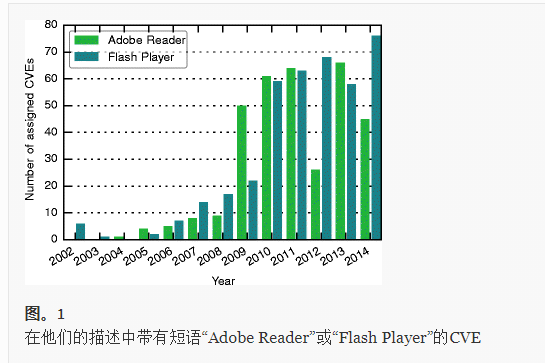
1. 3。

文档格式的灵活性和多功能性为混淆嵌入式恶意内容提供了充足的机会。

相同的功能也阻碍了恶意文件的识别并增加了检测工具的计算负担。

攻击者使用的最喜欢的格式是PDF（针对ADOBE READER），闪存（瞄准的Adobe Flash Player）和Microsoft Office文件[ [1](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR1)，[2](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR2) ]。在2012年，开创性的漏洞攻击工具Blackhole专门针对Java，PDF和Flash文件，其继任者继续这种做法[ [3](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR3) ]。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[ [2](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR2) ]。近期Flash已经广泛部署恶意广告，即通过广告网络在合法网站上放置恶意软件。甚至一些最着名的网站也成为这种攻击的受害者[ [3]](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR3)]。尽管普遍用于重定向到提供漏洞工具包的站点，但Flash文件直接定位到Flash Player并不罕见。

非可执行文件作为有*针对性的攻击*手段尤其受欢迎。近年来，针对政府和工业界的一系列高调针对性攻击，并且越来越普遍且越来越隐蔽。针对欧洲政府机构的Miniduke针对性攻击活动使用了利用Adobe Reader零日漏洞的复杂PDF文件。微软Office中的四个不同的零日漏洞被用于针对国防工业的Elderwood攻击。APT1或CommentCrew小组针对政府和行业目标在Adobe Reader和Microsoft Office中使用了0天漏洞[ [4](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR4) ]。在2014年发现的24个0天中，有16个是针对Adobe Reader和Flash Player的（参见图[1](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig1)），而微软的Word文件支配用于针对性攻击[文件类型列表[1](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR1)，[5](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR5) ]。在2015年的前9个月，漏洞利用工具包利用的前10个漏洞中有8个被报告为Flash Player漏洞[ [6](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR6) ]。



检测恶意非可执行文件的主要困难是理解复杂格式的必要性。虽然这种困难在基于动态分析的方法中被边缘化，即在仪表化的沙箱中渲染文件，但这些方法通常相当缓慢。静态分析方法以其高性能而闻名，通常会部署临时特定于格式的检测技术，这些检测技术并未在各种格式中进行概括。为了缓解这个问题，我们提出了一种*新的静态分析方法*，它有可能在各种格式中更具可移植性。我们的实验证明，通过合并适当的格式解析器，它可以应用于PDF和Flash文件。在介绍该方法的主要特点之前，我们回顾一下相关的工作。

1.1相关工作

上PDF恶意软件检测的早期工作集中在n-gram中的分析[ [7](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR7)，[8](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR8) ]的PDF文件在磁盘上。但是，PDF是一种复杂的文件格式[ [9](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR9) ]。PDF文件，尤其是恶意文件，通常以压缩形式使用*混淆*，使用不同的编码甚至加密。因此，只有完整的PDF解析器才能正确解除混淆。在这方面，n-gram方法过于简单。第一种使用PDF解析器的方法是PJScan [ [10](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR10)]。它基于嵌入在PDF文件中的JavaScript代码的词法属性使用异常检测。但是，它无法处理在运行时加载的JavaScript代码或不首先使用JavaScript的恶意软件。随后提出了两种简单的基于学习的方法，Malware Slayer [ [11](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR11) ]和PDFrate [ [12](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR12) ]，均使用基于PDF文件原始字节的启发式特征。目前提出的所有方法通常都称为*静态方法，*因为它们不会执行或模拟PDF文件的任何部分。根据他们对PDF文件的解析是否符合PDF标准[ [9]](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR9)，他们可以分为深层和浅层两种方法]（深）或不（浅）。**PJScan是迄今为止唯一提出的深层方法**。浅层方法的一个常见弱点是PDF格式物理结构伪造的相对容易性，以PDFrate [ [13](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR13) ] 为例。*所有*纯静态方法的一个共同缺点是它们无法检测动态加载的威胁，例如，当分析的文件不包含攻击代码，而是通过网络或其他文件加载它时。

除了所描述的静态方法之外，还开发了*动态*方法来利用通过观察运行时打开PDF文件的效果获得的附加信息。不依赖检查PDF文件，动态方法不受PDF混淆和物理结构伪造的影响。早期的研究是基于软件仿真[ [14](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR14)，[15](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR15) ]。然而，软件仿真表明易于逃避和计算密集型。其他流行的动态方法包括Wepawet [ [16](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR16) ]基于沙箱JSand [ [17](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR17) ]和基于CWSandbox [ [18](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR18) ]的MalOffice 。Snow等人 提出采用硬件虚拟化并评估其系统ShellOS [[19](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR19) ]在PDF恶意软件。Tang等人 在低级硬件功能上使用异常检测[[20](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR20) ]。虽然动态方法比静态方法更为准确，但它们的执行时间使其不适合实时检测繁忙网络上的恶意文档。此外，构建和维护动态检测器能够模拟每个版本的易受攻击的软件产品，并结合每个支持的操作系统和库的每个版本，这是一项成本高昂且技术上具有挑战性的任务。另一方面，从检测器中省略目标软件的一个组合就足够了，并且为该特定版本设计的威胁将不会被检测到。

为了达到静态方法的速度*和动态*方法的精度，随后开发了*静态和动态组合*方法。MDScan执行静态JavaScript提取和动态代码执行[ [21](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR21) ]，但使用未记录功能模拟PDF JavaScript API的复杂性阻止了完整且无错误的解决方案。另一方面，MPScan可以挂载到Adobe Reader中以获得完美的JavaScript提取和反混淆处理，但是会执行静态漏洞检测[ [22](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR22) ]。由于其设计，它仅适用于单个版本的Adobe Reader上的恶意软件检测，其动态组件需要几秒钟才能运行。

与迄今为止提出的全自动方法相比，Nissim等人 建议使用*主动学习*方法，其中人类专家手动标记机器学习算法的有趣样本，目标是使检测器保持最新的威胁[ [23](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR23) ]。他们概述了一种结合了签名检测和目前描述的多种方法的设计，但将其实施和评估留给未来的工作。有关许多提及的PDF恶意软件检测方法的更详细的调查，我们请读者参考[ [23](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR23) ]。

与PDF相比，近年来仅有两种方法提供了有关检测Flash恶意软件的研究。OdoSwiff系统从2009年开始使用基于启发式的方法对静态和动态分析获得的特征[ [24](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR24) ]。它于2012年由FlashDetect成功实现，它将检测从ActionScript 2升级到ActionScript 3漏洞，并用朴素贝叶斯分类器取代了基于阈值的方法[ [25](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR25) ]。这两种方法都基于经验方法，努力编码领域专家（即恶意软件分析师）关于SWF开发现有方式的知识。这些*专家功能*表现非常好。例如，FlashDetect的机器学习分类器使用仅包含每个类别的47个样本的训练数据集进行评估，但即使这个小样本大小也足以实现高检测精度。然而，正如作者指出的那样，一些基于启发式的特征对于承诺的逃避者来说并不健全。此外，嵌入式恶意软件可能会根据其与Adobe Flash Player的差异来检测所采用的动态执行环境，并将其行为视为一种反应。相反，这里提出的方法使用*数据驱动的方法*而不是专家功能，其检测基于良性和恶意SWF文件之间的结构差异。通过保持与漏洞无关的特性，它仍然对新型攻击持开放态度，其静态方法可以加快执行速度，并且不会受到运行时间的影响。

1.2贡献

所提出的检测方法基于对分级文档结构的分析，并且此后缩写为Hidost。它是斯坦迪奇和拉斯科夫在[ [26](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR26) ]中发表的以前着作的延伸，在此被称为SL2013。SL2013中引入的新颖性是使用*逻辑结构*来表征恶意和良性PDF文件。PDF逻辑结构是由PDF标准定义的高级构造，它将基本PDF构建块组织为功能性文档。结果发表在[ [26](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR26)]显示恶意文件的属性，如JavaScript的存在和最小限度的良性内容的使用可以从其逻辑结构准确地确定。作为一种深度静态方法，SL2013受到困扰浅层方法的PDF混淆和物理结构伪造的影响较小。根据包含66万PDF文件的真实世界的数据集进行评估，SL2013展示了检测性能和吞吐量的组合，在防病毒引擎和已发布的科研工作中保持无与伦比。然而，在时间戳数据的实际滑动窗口实验中，SL2013的检测性能表现出不一致。

Hidost继承了SL2013的所有优点。它在PDF文件中保持了近乎完美的检测性能和高吞吐量，该文件专门针对繁忙网络上的集中部署SL2013进行了量身定制。作为深度静态方法的进一步优势，Hidost不受PDF混淆和物理结构伪造的影响。

此外，Hidost解决了我们稍后发现的SL2013的某些缺点。特别是，我们开发了*结构路径整合*（SPC），这是一种用于合并相似特征的技术。这样的合并特征可以更好地保留逻辑结构的语义并减少特征集对特定数据集的依赖性。SPC的好处有三个：（a）逃避的攻击面减少; （b）随时间变化的特征集是有限的; 和（c）功能的数量急剧减少。总之，这些改进使得Hidost比SL2013更安全和实用。

然而，最重要的是，本文介绍了一种新颖的Hidost系统设计，使其能够推广到多种不相关的文件格式。据我们所知，Hidost是第一个适用于多种文件格式的基于静态机器学习的恶意软件检测系统。它的通用性是通过将基于PDF逻辑结构的特征定义扩展为具有层次逻辑结构（Flash的SWF格式）的第二种文件格式实现的。最后，更进一步，Hidost不仅考虑了文件的逻辑结构，而且还考虑了其​​内容，使得对SWF等区别度较小的格式具有更高的精度。

为了演示Hidost的出色检测性能，我们通过实验评估了两种格式：PDF和SWF。我们的评估协议旨在模拟数据驱动的检测方法的实际部署，并考虑到恶意数据的自然演变。在我们的协议中，检测模型在固定大小的数据窗口上进行训练，并部署一段时间。一旦模型被认为太旧，就会在另一个更新近的数据窗口上重新训练，并再次在有限的时间段内进行评估。与机器学习算法评估中常用的经典交叉验证方法不同，我们的实验性协议考虑了安全应用程序中数据的时间性质，并且从未预测过去的数据。

总之，本文的主要贡献如下：

* 基于静态机器学习的恶意软件检测器Hidost，这是第一个基于逻辑结构和内容适用于不同文件格式的系统。
* Hidost对PDF和SWF两种格式进行了实验性评估，旨在反映恶意软件检测器的操作环境，该文件在440,000个PDF文件和前所未有的40,000个SWF文件的数据集上执行。在我们的评估中，Hidost在PDF上的性能优于VirusTotal上的所有防病毒引擎，并在SWF文件中名列前茅。
* Hidost针对两种文件格式（PDF和SWF）的原型实现，作为开源软件发布。
* 复制这项工作所需的源代码，包括作为开源软件发布的实验和绘图。
* 数据集需要重现这项工作。

1.3大纲

本文的结构如下。Hidost适用的文件格式（即分层结构的文件格式）将在第[2](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec5)节中进行介绍，同时还将详细介绍PDF和SWF的逻辑结构。第[3](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec8)节介绍了Hidost的系统设计，其中包括从PDF和SWF格式中提取结构元素，特征定义，选择和压缩以及学习和分类。第[4](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec16)节介绍了实验评估，包括数据集和实验方案的描述以及结果的讨论。我们将讨论Hidost对其他文件格式的扩展，并在第[5](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec25)部分中介绍了其应用程序的概念设计到Office文件格式OOXML和ODF。最后，第[6](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec26)节给出了结论并概述了未来工作的未决问题。

2分层结构的文件格式

文件格式是作为存储某些信息的物理表示的手段而开发的。某些格式（例如文本文件）没有任何逻辑结构，但其他格式（例如HTML）却可以。HTML文件是HTML *元素*之间逻辑关系的物理表示。如在图的例子中[2只](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig2)显示了，在一个HTML文件中，一个*p*元件可能是的后代*体*元件，其又具有*HTML*元素作为它的父。



HTML元素具有层次结构形式的逻辑结构。本文中介绍的工作涉及以*分层结构的文件格式*检测恶意软件。文件格式的物理布局可能会大大偏离其逻辑布局，与所提出的方法的操作无关。分层结构的文件格式的例子包括：

* 便携式文档格式（PDF）
* SWF文件格式（SWF）
* 可扩展标记语言（XML）
* 超文本标记语言（HTML）
* 打开文档格式（ODF），一种基于XML的办公文档格式
* Office Open XML（OOXML），一种用于办公文档的基于XML的不同格式
* 可缩放矢量图形（SVG），一种基于XML的矢量图形格式

在下文中，我们描述了在Hidost，PDF和SWF中实现的两种文件格式的层次逻辑结构。

2.1可移植文档格式（PDF）

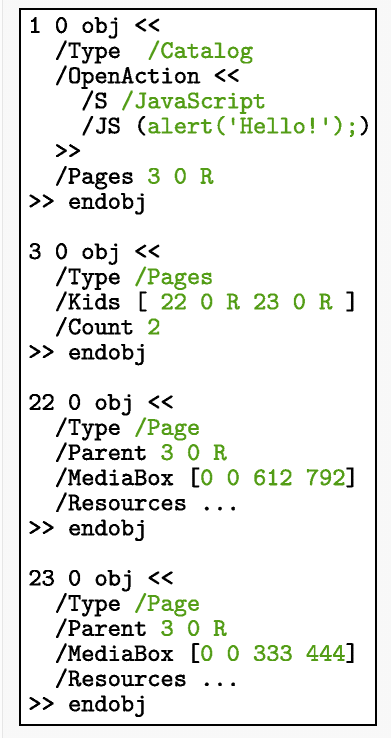
本部分被复制（与适应）从[ [26](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR26) ]，由互联网协会版权。

*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的语法包含以下四个主要元素：

* 对象。这些是PDF中的基本构建块。
* 文件结构。它指定了如何在PDF文件中布局和修改对象。
* 文档结构。它决定了对象在逻辑上如何组织来表示PDF文件（文本，图形等）的内容。
* 内容流。它们提供了一种手段来有效存储文件内容的各个部分。

PDF中有九种基本对象类型。简单的对象类型是布尔值，数字，字符串和空值。PDF字符串具有有限长度，并被括在括号“（”和“）”中。Name类型用作PDF文档结构描述中的标识符。名称使用字符“/”引入，可以包含除*空*（0×00）以外的任意字符。上述五种对象类型在本文中将被称为*基本*类型。数组是用方括号“[”和“]”括起来的PDF对象的一维有序集合。数组可能包含不同类型的PDF对象，包括嵌套数组。字典是在符号“<<”和“>>”之间包含的一组无用的键值对。钥匙必须是*名称对象，*并且在字典中必须是唯一的。这些值可以是任何PDF对象类型，包括嵌套字典。一个Stream对象是一个PDF字典，后跟一个字节序列。字节表示可以被压缩或加密的信息，并且关联的字典包含关于是否以及如何解码字节的信息。这些字节通常包含要呈现的内容，但也可能包含一组其他对象。最后，一个间接对象是以前定义的任何一个对象，这些对象提供了一个唯一的对象标识符，并包含在关键字obj和endobj中。由于其独特的标识符，间接对象可以通过*间接引用*从其他对象*引用*。

PDF对象的语法在图[3](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig3)所示的简化示例性PDF文件中示出。它包含由两部分对象标识符表示的四个间接对象，例如，第一个对象为1 0，以及obj和endobj关键字。这些对象是字典，因为它们被符号“<<”和“>>”包围。第一个是*Catalog*字典，由其*Type类型*条目表示，其中包含一个带有值*Catalog*的PDF名称。该目录有两个额外的字典条目：*页面*和*OpenAction*。*OpenAction*是一个嵌套字典的例子。它有两个条目：*S*，表示这是JavaScript动作字典的PDF名称，以及包含要执行的实际JavaScript脚本的PDF字符串*JS*：alert（'Hello！'）; 。*Pages*是具有对象标识符3 0的对象的间接引用：紧跟在Catalog之后的Pages字典。它有一个整数*Count*，表示文档中有两个页面，一个数组*Kids*可以用方括号标识，有两个对Page对象的引用。使用相同的对象类型来构建剩余的页面对象。请注意，每个Page对象都包含对*父*对象中Pages对象的反向引用条目。总而言之，有三个引用指向相同的间接对象，3 0，Pages对象。



各种基本对象之间的关系构成了PDF文件的逻辑，树状*文档结构*。文档结构中的节点本身就是对象，边对应于子对象驻留在父对象下的名称。对于数组，父 - 子关系是无名称的，并且对应于各个元素的整数索引。严格地说，文档结构不是树，而是一个定向的有根循环图，因为间接引用可能指向文档结构中任何位置的其他对象。这个图可以简化为一棵合适的树，称为*结构树*，将在[3.4](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec14)节中详细说明，因此我们将限制自己以简化的树形式考虑PDF文档结构，如图[4](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig4)所示。

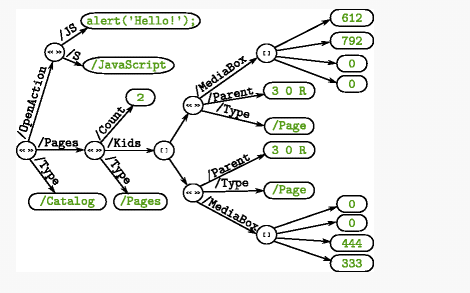


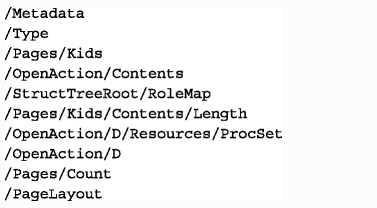
图4

PDF文件的结构树如图[3](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig3)所示。字典使用符号“<< >>”来表示，数组使用“ *方括号* ”。为简单起见，省略了周期

文档结构中的根节点是一个特殊的PDF字典，其中包含名称*Catalog*的强制*Type类型*条目。任何基本类型的对象都构成文档结构中的一个叶子，即终端节点。

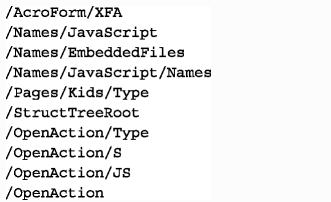
我们在PDF结构树中定义一条*路径*，作为从目录字典开始并以基本类型的对象结束的一系列边。例如，在图[4中，](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig4)存在从根（即，最左边的节点）到名为/ Pages和/ Count的边缘到值为2的终端节点的路径。这种PDF文档结构中的路径定义我们表示*PDF结构路径*，在我们的方法中起着核心作用。我们将路径打印为从根节点开始到叶节点结束时在路径遍历期间遇到的所有边标签的序列。我们之前例子的路径将被打印为/ Pages / Count。

以下列表显示了来自真实世界良性PDF文件的示例性结构路径：



我们的调查显示，这些结构路径在文件中的存在最能表明文件是良性的，或者缺少文件表明文件是恶意的。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 /伯爵。

以下是来自真实世界恶意PDF文件的结构路径列表：



2.2 SWF文件格式

*SWF文件格式*（*SWF*，发音为*swiff*）是一种专有的二进制文件格式，其规范在[ [27](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR27) ]中发布。SWF文件包含一个标题和一系列标记，即具有预定义字段值的数据结构。指定了65种不同类型的标签，每种标签都定义了自己的一组具有不同名称和数据类型的字段。一些基本的SWF数据类型是[ [27](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR27) ]：

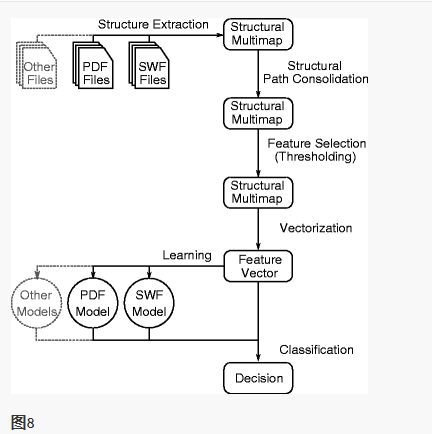
* 8位，16位，32位和64位整数，包括有符号和无符号整数，这些类型的数组和整数的可变字节数
* 不同宽度和精度的固定点和浮点数
* 宽度不是2的整数和定点数字
* 字符串
* 数据结构，如24位和32位彩色记录，矩形记录，2D变换矩阵等

图[5](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig5)显示了一个用于说明目的的非常小的SWF文件。显然，SWF的物理布局对于直接解释来说太模糊了。相反，我们对SWF逻辑结构的描述是基于对同一文件进行解码的，人类可读的描述，如图[6](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig6)所示。原始SWF文件的文本描述是使用SWFRETools工具包[ [28](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR28) ]中的ConsoleDumper类生成的，该工具包是用于反向工程SWF文件的开源Java工具包。

3系统设计

Hidost被设计为恶意软件检测系统，能够学习根据逻辑结构区分恶意文件和良性文件。由于各种文件格式的语义异质性，很难想象单一格式可以作为所有可能的分层结构文件格式的“共同标准”。然而，我们的设计明确地将格式特定的处理步骤与检测方法分开。因此，我们目前在PDF和SWF格式中测试的方法可以通过实现特定于格式的组件来扩展到其他格式，而无需重新构建其通用框架。所提出的方法是作为一个研究原型来实现的，其特征提取子系统被公开为开源软件[ [29]](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR29)]。已发布的代码包含一个用于从PDF（用C ++实现）和SWF文件（用Python和Java实现）提取功能的工具集。实验复制代码将单独发布，如第[4](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec16)节所述。

Hidost的系统设计如图[8所示](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig8)。**Hidost有六个主要阶段：结构提取，结构路径合并，特征选择，矢量化，学习和分类。**结构抽取将特定格式的结构特征转换为表示结构层次结构中的路径的公共数据结构 - 结构化多图。结构路径合并旨在将结构路径转换为更一般的形式，消除工件。特征选择涉及找到成功的机器学习应用程序所需的最小特征集。矢量化将结构化多图表转换为机器学习方法处理的数字矢量。学习根据编码在特征向量中的属性生成恶意和良性文件的区分模型。最后，



3.1文件结构提取

我们方法的第一步是将文件转换为更抽象的表示形式，即逻辑结构。这一步对我们的方法至关重要，因为它实现了两个关键目标：（a）使用逻辑结构区分恶意文件和良性文件，以及（b）适用于多种文件格式。

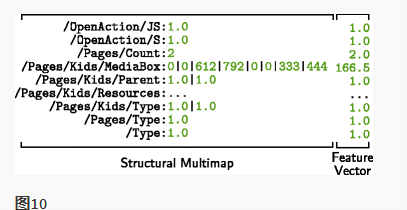
分层结构文件格式的逻辑结构的适当表示是*结构化多图*。多图是普通地图数据结构的一般化，也称为字典或关联数组。尽管映射提供了键与对应值之间的映射，但multimaps将键映射到一*组值*。结构化multimap是一个multimap，它将结构树的每个结构路径映射到位于给定路径上的所有叶子集合。在地图术语中，结构路径代表路径所映射的所有叶子的*键*和集合，以表示地图的*值*。图[9中](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig9)示出了示例结构化多图。使用竖线符号“|”分隔同一个键的多个值。Hidost使用本节后面介绍的简化形式的结构化多图。

**3.1.1 PDF**

PDF逻辑结构被组织为定向的有根循环图。为了将其转换为结构化多图，首先需要将图减少为定向的有根树。这是通过从图中删除所有循环来实现的。当来自深度*d R*处的树节点的间接引用指向位于深度*d T* < *d R*处的相同路径上的树节点时，PDF逻辑结构中存在周期。图[3](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig3)显示了一个示例：深度为3 的路径/ Pages / Kids / Parent上的间接引用指向位于/ Pages处的字典 深度为1，位于同一条路径上。间接引用也可能导致提取的树结构不一致。当位于不同路径的两个间接引用引用同一个对象时，它是不明确的，这是“真实”的。

通过基于广度优先搜索（BFS）算法的简单程序隐含地解决了所描述的两个问题 - 结构图中的循环和多个参考 - 。需要一个强大的PDF解析器来导航混淆或格式不正确的PDF文件的PDF结构。Hidost利用开源的Poppler PDF渲染库[ [30](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR30) ]，版本为0.18.4。该过程首先查找文件结构的根节点，即*目录*字典。然后，它在整个文件结构图上执行广度优先搜索，插入所有对（*p，l*），其中*p*是结构路径，*l*是位于路径*p*上的叶节点，结果到结构化多图中。通过跳过指向先前访问的对象的间接引用并将它们视为叶节点来避免循环。按照字母顺序的子节点枚举确保多个引用在同一节点上的一致解析，以便每次遍历PDF文件的结构图都会生成相同的结构化多图。

更确切地说，Hidost并没有使用完整的结构化多图，而是使用了简化。这种简化涉及非数字数据类型的处理，即除整数，实数和布尔值之外的所有类型。字符串，PDF名称和其他非数字类型（都可转换为字符串）被替换为常量值1.非数值的二值化可以通过比较图1和图2来看出。[9](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig9)和[10](https://link.springer.com/article/10.1186/s13635-016-0045-0#Fig10)。这种治疗的选择是完全放弃非数字值和进行广泛评估之间的折中。不同的方法来串类数据类型的治疗中已经提出了相关工作，从静态的，例如，在度量空间[中嵌入字符串[10](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR10)，[31](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR31)]，用简单的性质，例如其长度，熵，或关键字[分布表征他们[11](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR11)，[12](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR12) ]或测试它们用于有效的CPU指令[ [15](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR15) ]，以动态的，例如，字符串[的CPU仿真[14](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR14)，[21](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR21) ]化合物或它们执行[ [19](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR19)，[20](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR20)，[22](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR22) ]。然而，所有这些方法要么与所需的静态系统设计（动态评估）相冲突，要么降低计算性能（字符串嵌入和CPU指令测试），要么容易被忽略（简单字符串属性）。



实验证实了使用二值化非数字值与完全省略相比的积极影响。第[4.3.4](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec23)节评估了从SL2013使用的纯二元特征迁移到Hidost数值的影响。

3.2结构路径合并

许多文件格式的语法丰富性和灵活性使得语义等价但语法上不同的结构成为可能。这种语法多态性可能会降低检测的准确性，并且还会使攻击者完全避免检测特定文件。为了解决这个问题，我们开发了一种启发式技术，用于合并结构路径，从而将多态路径缩减为一致的表示。这种技术可以作为PDF格式的最佳例证。

我们已经观察到，PDF文件之间通用的许多路径展现出与其他路径的结构相似性。事实上，我们能够识别彼此相似但不完全相同的路径组。这些群体中的路径以两种方式之一表现出相似性。除了一个可自定义的路径组件外，某些组中的路径是相同的，而其他组中的路径共享一个共同的重复子路径。然而，重要的是，一组相似路径中的所有路径指的是PDF中具有相同目的的对象，即相同的*语义含义*。例如，路径/ Pages / Kids / Resources和/ Pages / Kids / Kids / Resources有一个共同的重复子路径，/ Kids，但都是指具有相同目的的PDF字典 - 为呈现PDF文件页面所需的资源提供名称。从语义上讲，结构提取算法在访问页面的资源字典之前采用哪条路径是无关紧要的 - 所有资源字典都具有相同的语义含义。同样，路径/ Pages / Kids / Resources / Font / F1和/ Pages / Kids / Resources / Font / F42仅在最后一个路径段中有所不同，PDF标准要求用户定义这两个路径段，但都涉及字体词典描述用于PDF文件的字体。同样，无论具体的PDF编写器赋予字体字典具体的名称，所有的字体字典在语义上都是相同的。

语义上等价的路径组存在着对SL2013特征定义的效用的问题，SL2013特征定义将每条路径视为一个独特的特征。通过将所有语义等同的路径合并到一个特征来保留路径的语义更有意义。这种被称为*结构路径整合*（SPC）的思想是在Hidost中实现的，并在[4.3.3](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec22)节中进行了实验评估。

SPC的实现基于使用正则表达式替换关键路径组件。重复的子路径将从路径中完全删除。例如，以上作为具有共同重复子路径的示例所指示的两个路径将被合并到路径/页面/资源中，去除重复子路径/孩子。另一方面，用户定义的路径组件是*匿名的*，即用占位符路径组件/名称替换。例如，上面例子中用户定义字体名称的两个路径将合并到路径/ Pages / Kids / Resources / Name中（如果关于重复路径的规则当然没有事先应用）。表[1](https://link.springer.com/article/10.1186/s13635-016-0045-0#Tab1)列出了Hidost for PDF中采用的SPC规则，使用BOOST.REGEX库实现。每个规则包含两个正则表达式：一个用于搜索要替换的模式（左），另一个用于确定替换字符串（右）。

。特别是对PDF而言，我们有充足的机会进一步匿名化和层级化，例如我们的规则中未涉及的名称树和数字树。通常，为了扩展列表，建议阅读PDF标准，查找PDF文档结构中允许使用用户定义名称或有明确定义的列表或层次结构的位置。然而，与SL2013相比，即使这套有限的规则也提供了以下关键优势：

* 减少攻击面。如果没有SPC，出现次数高于阈值的每条不同路径都会构成一个特征。在这种情况下，努力逃避检测的攻击者可能执行*隐藏攻击*通过在与功能集中的任何路径不同的自定义路径上隐藏恶意负载。例如，具有长随机生成名称的字体的路径很可能以前不会遇到。插入其中的恶意有效载荷对于在其功能集中没有此特定路径的检测器是不可见的。对于不允许使用用户定义路径的SWF，有效载荷可能隐藏在非常深的层次结构中，而不是在“正常”文件中遇到。PDF也受此漏洞的影响。隐藏攻击实施便宜，是SL2013的主要弱点。Hidost利用统一路径关闭了逃避途径。
* 有限的功能集在时间上漂移。在现实世界的机器学习应用中，手头的问题经常会随时间而改变。在安全应用中尤其如此，在这种应用中，防御者被迫适应不可预测的攻击变化。这个问题在机器学习中被称为*概念漂移* [ [32](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR32) ]，并且最近开始引起对安全文献的兴趣[ [33](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR33) ]。

随着时间的推移，数据的不断变化使得分类器越来越过时。因此，需要定期更新学习模型，即定期分类器再训练。在这项工作中，与数据相关的特性，建议在每次再培训前重新进行特征选择，以便更好地适应概念漂移。定期特征选择会导致现有特征的过时以及在两个再训练期之间添加新特征。我们将由周期性特征选择引起的特征集中的变化称为*特征集漂移*。由于结构路径定义的灵活性，PDF比SWF更容易受到特征集漂移的影响。如[4.3.3](https://link.springer.com/article/10.1186/s13635-016-0045-0#Sec22)节所述 显示，SPC有效地用于减少Hidost中的特征集漂移。

* 特征空间降维。最后，SPC对功能的总数有巨大的影响。特征空间维度直接影响学习算法的运行时间和内存需求。在我们的定期再训练PDF实验中，平均特征集大小减少了88％，从每次训练的10,412.5到1237.4个特征。但是，对于手动制作的路径，SPC的有效性存在限制。由于它不知道语义上有效的路径，所以SPC无法处理不可预见的情况，例如目录字典中的任意名称。为了解决PDF逻辑结构覆盖范围中的这个最后的“盲点”，需要采用白名单方法，其完整和最新的表示方式 - 整个结构的模型 - 这超出了本文的范围。

减少的攻击面和有限的特征集漂移对Hidost作为基于机器学习的检测器的操作安全性起着重要作用。大量减少的功能数量使其能够应用于更大的数据集。所描述的改进结合在一起，使Hidost在真实世界的操作环境中作为精确，可靠和安全的恶意文件检测器的适用性迈出了一大步。

3.3功能选择

尽管通过结构路径合并减少了语法多态性，但观察数据中可能仍然存在很少发生的路径。使用这些路径构建区分模型会增加输入空间的维度，而不会提高分类准确性。因此，必须执行特征选择（在其他机器学习应用中常见），以限制罕见特征的影响。在介绍具体特征选择技术之前，我们讨论了本文详细研究的两种格式中罕见特征出现的原因。

SWF文件格式规范[ [27](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR27) ]严格定义所有标签及其所有字段的名称，禁止定制。因此，Hidost的SWF功能理论上包含由SWF规范定义的每个结构路径。但是，实际上，我们没有努力枚举SWF逻辑结构中的所有路径。相反，特征集合包括在训练数据集中*观察到的*所有路径，总共3177个。

相比之下，PDF文件格式规范[ [9](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR9) ]允许在任何PDF字典中使用用户定义的名称，从本质上启用无限数量的不同路径。我们的数据表明，这个PDF功能在实践中被广泛使用，因为我们已经观察到超过900万个不同的PDF结构路径。但是，23这些路径不会出现在多个文件中。发生在小数据集中的这些和其他路径被认为是异常的。因此，原SL2013方法选择了出现在固定数量的训练文件中的PDF路径，即其特征集合，即1000。该阈值控制检测精度（更多路径）和模型简单性（更少路径）之间的折衷，并且可以自由调整。

在SPC之后并且在我们的定期再训练实验方案中的每次训练之前，我们应用相同的出现阈值，即1000个文件，其对应于训练集大小的约1％。这与SL2013形成鲜明对比，SL2013在“事后”中进行特征选择，对整个数据集进行一次特征选择。

3.4矢量化

结构化多图是PDF和SWF逻辑文件结构的合适表示，但它们不能直接用于机器学习算法。它们首先需要被转换成特征向量，即*特征空间* R中的点ñ，在一个称为矢量化的过程中。

在矢量化过程中，结构化多图表首先被结构图 - 普通地图数据结构所替代，这些结构图将结构路径映射到相应的单个数值。为此，对应于多映射中的一个结构路径的每组值被减少到其中值。我们选择了中位数为高于平均值（这里，我们用这个词更强大的统计*稳健*在统计意义，表示此位提供了异常的情况下设定值的更好的表征，而不是它提供了对敌对逃避任何健壮性）。这条规则的唯一例外是，主要由布尔值组成的SWF结构化多元图中的值集合被简化为其平均值，而不是中值。均值保留有关布尔值的更多信息，而不是中值，它只能为0，12，或1，并且不存在异常值。由于其逻辑结构具有相对较少的布尔值，因此该例外不适用于PDF。

结构图被转换成特征向量f∈ [Rñ通过为每个特定的结构路径预留一个单独的维度，并使用结构图中的值作为特定维度的值。在特征提取之前和特征选择期间定义个体结构路径到特征向量维的映射，并且在训练和分类之前均匀地应用于所有结构图。因此，特定的结构路径对应于每个特征向量中的相同维度，使得学习算法能够理解特征向量。

.5学习和分类

到目前为止所呈现的阶段将样本（即，文件）转换成适合作为机器学习算法的输入的特征向量。具体机器学习分类器的选择取决于多个参数，例如，数据集大小，特征空间维度，可用计算资源，针对对抗攻击的鲁棒性等，并且分类器针对不同用途而量身定制。因此，已公布的Hidost实施不包括学习和分类子系统。相反，其输出可以与读者选择的分类器一起使用。对于本文中介绍的实验，开源的scikit-learn Python机器学习库的Random Forest实现[ [34]](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR34)]，版本0.15.0b2。作为实验性复制代码的一部分，Hidost的这部分单独发布，详见下一节。

随机森林[ [35](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR35) ]是一个整体分类器。它是通过使用CART方法生长决策树的森林来培训的。*t个* RF树中的每一个都是在它自己的固定尺寸随机子集上随着替换而绘制的训练数据。在训练期间在树的每个分支处，提供最优分割的特征从包含*f*RF的随机子集中选择以前不用于此树的功能。在分类过程中，每棵树的决策都被计为一票，总体结果是大多数选票。随机森林以其卓越的泛化能力和抗数据噪声的稳健性而闻名。对于实验评估，将森林大小设置为200棵树，并将其他所有参数设置为scikit-learn默认值。

6结论

在本文中，我们介绍了Hidost，一种基于机器学习的恶意软件检测系统。它代表了先前发表的方法SL2013 [ [26](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR26) ] 的扩展。Hidost是第一个基于静态机器学习的恶意软件检测器，旨在操作多种文件类型。它通过根据结构和内容制作恶意和良性样本模型来实现这一点。

在一个实际的实验中对实际数据集进行了评估，这些实验包括多个月的定期再训练，Hidost胜过了网站VirusTotal部署的所有防病毒引擎，并检测到最高数量的恶意PDF文件。它也在SWF恶意软件中名列前茅。与其前身SL2013相比，它不太容易受到隐藏在PDF文件模糊部分的恶意软件的影响。通过定期再训练，Hidost也变得更加强大，能够不断适应恶意软件，以更新防御。最后，其大大减少的特征集维度使其能够在非常大的数据集上高效地应用。

对于Hidost来说，合理的下一步是对其他分层结构文件格式的实施和评估。特别重要的是Microsoft Office使用的格式的应用程序，因为它们广泛用于最近的针对性攻击。本文提出了该应用的概念设计。展望未来，更高级的字符串处理方法的发展可能证明对于逻辑结构和数字内容不能提供足够区分能力的格式来检测恶意软件是不可或缺的。