**PlatPal：  利用平台多样性检测恶意文档**

**美国*佐治亚理工学院的* Meng Xu和Taesoo Kim**

  
https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/xu-meng

**这篇论文被收录在“论文集”中**

**第二十六届USENIX安全研讨会**

**2017年8月16 - 18日•加拿大不列颠哥伦比亚温哥华**

ISBN 978-1-931971-40-9

P LAT P AL ：检测与平台多样性恶意文件

Meng Xu和Taesoo Kim

*佐治亚理工学院*

抽象

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，例如解析器混淆和分类器规避攻击。

针对这一点，我们提出了一个新的视角maldoc检测：平台的多样性。具体来说，我们确定了操作系统设计和实现中可能导致行为分歧的八个因素，从系统调用语义（更明显）到堆对象元数据结构（更加微妙），并进一步展示如何阻止攻击者发现错误，利用错误或执行恶意活动。

我们进一步对P LAT P AL 进行原型系统收集平台多样性。P LAT P AL 挂接到Adobe Reader以跟踪内部PDF处理，并使用沙箱执行来捕获maldoc对主机系统的影响。比较不同平台上的执行跟踪，并且maldoc检测基于良性文档在不同平台上表现相同的观察结果，而maldoc在开发过程中表现不同。评估表明，P LAT P AL 在良性样本中不会产生虚假警报，在恶意样本中检测到各种行为差异，是一个可扩展和实用的解决方案。

1        引言

网络攻击者正在转向基于文件的恶意软件，因为用户聪明起来恶意电子邮件附件和网络链接，许多防病毒（AV）供应商[如建议39 ，50 ，54 ，57 ]。用户通常会被浏览器，电子邮件代理或AV产品更多地警告可执行文件的危险，而像PDF这样的文档由于受到静态文件的影响而受到的关注和审查要少得多，并且几乎没有什么危害。

但是，随着时间的推移，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。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。

一个maldoc通常利用其解释器中的一个或多个漏洞发起攻击。幸运的是（或者不幸的是），鉴于文档阅读器越来越复杂以及库/系统组件的依赖性越来越大，攻击者的攻击面越来越大。新的漏洞继续被发现，在2015年发布了137个CVE，在2016年仅有Adobe Acrobat Reader（AAR）。AAR的普及及其大型攻击面使其成为攻击者的首要攻击目标[ 25 ]，其次是浏览器和操作系统内核。在引入了类似Chrome的沙盒机制后[ 2 ]，在pwn2own竞赛中，单个漏洞的价值可能高达7万美元[ 21]]。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[ 37 ]，字体管理器[ 28 ]以及JavaScript引擎[ 22 ]。系统范围的依赖性，如图形库[ 23 ]也在攻击者的雷达。

随着PDF格式的普及，AAR的持续开发使得马尔科夫检测成为一个紧迫的问题，并且近年来已经提出了许多解决方案来检测具有恶意有效载荷的文档。这些技术可以分为两大类：静态和动态分析。

静态分析，或基于签名的检测[ 14 ，27 ，31 ，33 ，34 ，36 ，46 ，52 ，59 ]，解析的恶意内容的指示，如shellcode的或相似性与已知的恶意软件样本文档和搜索。在另一方面，动态分析，或基于执行的检测[ 45 ，48 ，58 ]，运行的部分或整个文档和迹线的恶意行为，如脆弱API调用或返回导向编程（ROP）。

然而，最近的作品突出了这些解决方案的一些共同的缺点。Carmony *等*。[ 11 ]表明，在这些解决方案中使用的PDF解析器可能已经过分简化了关于PDF规范的假设，导致恶意负载的不完整提取和分析失败。也已经证明，基于机器学习的检测有可能以原理和自动的方式被回避[ 35 ，

53 ，65 ]。另外，许多解决方案只关注JavaScript部分，而忽视了与其他PDF组件在发起攻击方面的协同作用。因此，尽管现代AV产品支持PDF漏洞检测，但即使后者仅对现有漏洞进行微小修改，也不能迅速适应新颖的模糊技术[ 55 ]。由于缺乏攻击程序和运行时间跟踪，AV产品还会出现提供零日攻击保护的问题。

在本文中，我们提出了一个Maldoc检测方案P LAT P AL ，用于分析不同平台（例如Windows或Macintosh（Mac））上恶意文档文件的行为差异。与依赖现有恶意软件样本构建启发式的静态和动态检测方案不同，P LAT P AL 基于一套完全不同的见解：1）良性文档在不同平台上表现相同（在一定水平上），而2 ）恶意文档在不同平台上启动漏洞时会导致不同的行为。

第一个假设可以通过打开许多使用跨平台的各种PDF功能的良性样本进行实证验证。为了支持第二个假设，我们调查了OS受到攻击时可能导致行为分歧的因素，并从系统调用语义（更明显）到堆对象元数据结构（更精细）等8个因素中找到了这些因素。我们进一步展示了如何利用它们阻止攻击者发现错误，利用错误或执行恶意活动。

P LAT P AL 基于这些见解。要检测文档是否具有恶意负载，P LAT P AL 将使用相同版本的AAR实例打开它，但在不同的操作系统上运行。P LAT P AL 在处理文档时记录AAR的运行时跟踪，并随后在各个平台上进行比较。执行痕迹和产出的共识表明文件的健康，而分歧则表示攻击。

虽然这个过程听起来简单直观，但是需要解决两个实际的问题，以使得P LAT P AL 工作：1）在不同的平台上，“行为”可能有什么不同？2）他们如何被普遍追踪？P LAT P AL 追踪并比较了两种类型的行为。内部行为包括AAR在PDF处理周期中执行的关键功能，如加载，解析，呈现和脚本执行。外部行为包括文件系统操作，网络活动和程序启动。这与典型的恶意软件分析工具，如杜鹃沙箱[ 44 ]。

值得强调的是，P LAT P AL 不应被视为当前的恶意软件分析工具（如Cuckoo [ 44 ]）的竞争对手，因为它们依赖于不同的假设。目前的工具在很大程度上依赖于OS范围的活动的黑名单（或白名单）的可用性，因此可以对样本的行为进行审查。这种方法适用于已知的恶意软件，但可能会在0天的PDF漏洞攻击中丧失优势。另一方面，P LAT P AL 不要求这样的列表起作用，而仅仅依赖于这样一个事实，即攻击者很难在Windows和Mac平台上以完全相同的方式制作一个利用AAR的恶意PDF 。

P LAT P AL 是针对使用PDF规范中的各种特征的1030个良性样本进行评估，并且报告没有差异，即没有误报。对于利用16个不同的CVE的320个maldoc样本的集合，P LAT P AL 可以检测其中的209个发散和另外34个样本使两个AAR实例崩溃。其余部分由于各种原因而未被发现，例如针对旧的和特定版本的AAR或未触发恶意活动。P LAT P AL 可以在不超过24秒的时间内完成文件的扫描，不需要手动驱动。

纸张贡献。总之，本文做出如下贡献：

•    我们建议跨不同平台执行文档，并使用行为差异作为maldoc检测的指标。

•    我们对平台多样性进行深入分析和分类，并展示它们如何用于检测Maldoc攻击。

•    我们基于这些见解对P LAT P AL进行了原型设计。评估证明P LAT P AL 是可扩展的，不会引起虚假警报，并检测恶意样本中的各种行为差异。

我们计划开源P LAT P AL，以提示使用平台多样性进行Maldoc检测，并启动一个PDF maldoc扫描服务供公众使用。

2        马尔多克检测：调查

现有的maldoc检测方法可以大致分为两类：1）动态分析，在特定的仪器环境中执行和检测恶意代码; 和2）静态分析，其中执行检测而不执行代码。表1 列出了现有方法的总结。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image006.gif  E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image006.gif   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 类别 | 焦点 | 检测技术 | 解析器？ | ML？ | 模式？ | 逃避/缺点 | | 静态的 | JavaScript的  JavaScript的  JavaScript的  JavaScript的 | 词汇分析[ 27 ]  令牌集群[ 59 ]  API参考分类[ 14 ]  Shellcode和操作码签名[ 31 ] | 是  是  是  没有 | 是  是  是  没有 | 是  是  是  是 | 沉重的混淆，  代码加载 | |  | 元数据 | 线性化的物体路径[ 36 ] | 是 | 是 | 是 |  | |  | 元数据  元数据 | 分层结构[ 33 ，52 ]  内容元功能[ 46 ] | 是  是 | 是  是 | 是  是 | 模仿[ 53 ]，  反向模仿[ 35 ] | |  | 都 | 很多上面提到的启发式[ 34 ] | 是 | 是 | 是 |  | | 动态 | JavaScript的  JavaScript的  JavaScript的 | Shellcode和操作码签名[ 58 ]  已知的攻击模式[ 45 ]  内存访问模式[ 48 ] | 是  是  是 | 没有  没有  没有 | 是  是  是 | 不兼容的JS引擎，  基于非脚本的攻击 | |  | JavaScript的 | 常见的maldoc行为[ 29 ] | 没有 | 没有 | 是 | 零日攻击 | |  | 文件 | 违反内存访问不变量[ 62 ] | 没有 | 没有 | 没有 | ROP和JIT喷雾 |   E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image007.gifE:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image008.gifE:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image007.gifE:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image009.gif表1：恶意PDF文档检测技术的分类。这种分类方法部分基于系统调查报告[ 40 ]，增加2013年以后的作品以及摘要分析器，机器学习和模式依赖关系， |

规避技巧。

2.1        静态技术

静态分析工作的一个重点在于JavaScript内容的开发重要性，例如，大多数（根据[ 58 ]的超过90％）maldocs使用JavaScript来完成攻击。PJScan [ 27 ]依靠词汇编码风格，如变量名称，括号和运算符的数量来区分良性和恶意的JavaScript代码。Vatamanu *等*。[ 59 ]将JavaScript代码标记为变量类型，函数名称，运算符等，并将令牌集群构建为良性和恶意文档的签名。同样，Lux0r [ 14 ]分别构造了两组在良性和恶意文档中找到的API参考模式，并用它来对maldocs进行分类。MPScan [31 ]与其他JavaScript静态分析器的区别在于它挂钩了AAR并动态地提取JavaScript代码。但是，鉴于代码分析仍然是静态的，我们认为它是一种静态分析技术。

这些方法的一个共同的缺点是，它们可以通过沉重的模糊处理和动态代码加载（除了[ 31 ]，因为它在运行时挂钩到AAR中）。静态解析器根据JavaScript代码可以放置/隐藏的位置的预定义规则提取JavaScript。但是，考虑到PDF规范的灵活性，攻击者的创意隐藏了代码。

另一个工作重点是检查PDF文件元数据，而不是其实际内容。部分原因在于混淆技术倾向于滥用PDF规范中的灵活性，并通过改变正常的PDF结构来隐藏恶意代码。PDF恶意软件杀手[ 36 ]使用特定PDF元素（例如，/ JS ，/页等）的线性化路径来构建maldoc分类器。Srndic *等*。[ 52 ]和Maiorca *等*。[ 33 ]更进一步，也使用分层结构进行分类。PDFrate [ 46]包括另一组特征，例如字体的数量，流的平均长度等以改进检测。Maiorca *等*。[ 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。

所有的方法都是基于这样的假设：在混淆期间正常的PDF元素层次结构是失真的，并且创建了在良性文档中通常不存在的新路径。但是，这个假设受到两次攻击的挑战。Mimicus [ 53 ]通过在文档中添加空的结构和元数据项目来实现模仿攻击并修改现有的maldoc，使其看起来更像良性，而对渲染没有实际的影响。相反，反向模仿[ 35 ]攻击尝试将恶意内容嵌入到良性PDF中，同时尽可能少地修改恶意内容。

2.2        动态技术

所有被调查的动态分析技术都只关注嵌入式JavaScript代码，而不是整个文档。MDScan [ 58 ]在一个定制的SpiderMonkey解释器上执行提取的JavaScript代码，不断扫描解释器的存储空间寻找已知形式的shellcode或恶意操作码序列。PDF

Scrutinizer [ 45 ]通过钩住Rhino 解释器并扫描已知的恶意代码模式（如堆喷，shellcode和易受攻击的方法调用）采取了类似的方法。ShellOS [ 48 ]是一个轻量级的操作系统，用来运行JavaScript代码并记录它的内存访问模式。在执行期间，如果内存访问序列匹配已知的恶意模式（例如，ROP，关键系统调用或函数调用等），则该脚本被认为是恶意的。

尽管这些技术在检测恶意负载方面是准确的，但它们还是有一个共同的问题：一个不兼容的脚本环境。AAR的JavaScript引擎不仅遵循ECMA标准[ 18 ]，还遵循Acrobat PDF标准[ 1 ]（如Adobe DOM元素）。因此，如果没有仿真，像doc ，app ，甚至是这样的对象 （这在良性和恶意文档中都很常见）将无法正确运行。另外，恶意负载可以被编码为文档中的字体或图像对象[ 37]]，既不会被提取也不会被检测到。某些攻击也可能利用内存布局方面的知识，例如AAR及其相关库中存在的ROP小工具或功能，这在外部分析环境中很难模拟。

相反，模拟JavaScript的执行环境，刘*等人*。[ 29 ]使用上下文监视代码来处理PDF文档，并使用AAR自己的运行时执行JavaScript代码，因此不受不兼容性问题的影响。但是，仪表代码只监视常见的和已知的恶意行为模式，如网络访问，堆喷，DLL注入等，这些都不是完全通用的，当新的恶意代码反检测措施出现时必须扩展向上。CWXDetector [ 62 ]提出的一个w ⊕ X状的方法来检测在执行期间由maldocs注射非法代码。但是类似于W⊕X，其有效性在ROP和JIT-喷涂的情况下受到损害。

2.3        总结和动机

调查maldoc检测技术产生了一些有趣的观察：

解析器的依赖。由于文档由数据（例如文本）和可执行文件（例如脚本）组成，因此常见的模式是首先提取可执行的组件，并进一步用静态或动态分析来检查它们。为此，通常假设能够以与AAR相同的方式解析PDF文档的解析器。如表1 所示，除了三种方法外，其他所有方法都使用开源或自行开发的解析器，并假定其功能。然而，Carmony *等*。[ 11 ]表明，这些解析器通常是不完整的，并且在可以嵌入JavaScript的地方有过于简单的假设，因此可以启动*解析器混淆攻击*来轻易地逃避检测。

机器学习依赖。机器学习技术在maldoc检测中被大量使用，特别是在静态分析中，由于其在分类/聚类方面的能力，而没有事先了解该模式。如表1 所示，13种方法中有7种使用机器学习来区分良性和恶意文档，另外四种方法也可以转换为使用机器学习进行启发式挖掘。然而，最近提出的对抗性机器学习技术[ 20 ，42 ，65 ]提高对基于在对手的存在表面特征分类的有效性产生严重怀疑。例如，徐*等人*。[ 65]能够以类似于遗传编程的方式自动产生不具有关于分类器的知识的回避maldoc变体。

结构/行为差异。在调查的方法中隐含的假设是良性文件和恶意文件之间存在结构/行为差异，并且可以观察到这种差异。由于文件必须遵循公共格式规范，所以在良性文件中预期共同性（结构或行为）。如果一个文件很大程度上偏离了良性样本的规范或常见模式，则更可能是一个恶意文件。然而，这样的假设是由Mimicus [ 53 ]和反向模仿[ 35 ] 挑战的]攻击的方式，如果攻击者知道用于区分良性和恶意文档的模式，maldoc可以系统地避开检测。另外，导出差异模式需要已知的恶意软件样本。因此，表1中的所有方法都需要已知的恶意软件样本，以自动学习模式或基于启发式，期望或经验手动定义模式。这限制了他们检测零日攻击的能力，而这些攻击在没有获得任何先验知识的情况下得以实现

全面的动态分析。值得注意的是，只有一种动态检测方法对整个文件进行分析，相反，其余的方法只对提取的JavaScript代码执行分析。这与传统的沙盒式恶意软件分析（如Cuckoo [ 44 ]或CWSandbox [ 63 ]）形成鲜明对比，后者执行恶意软件，并在运行时检查其行为和对主机操作系统的影响。一个原因可能是因为maldoc运行在AAR之上，AAR本身是一个复杂的软件，在主机系统上留下了很大的空间。maldoc执行的痕迹隐藏在大的脚印中，使分析更加困难。

动机。P LAT P AL 的发展受到了马尔多克检测研究中上述问题的推动。我们设计P LAT P AL 以：1）与预期的翻译人员（即本文中的AAR）分享相同的文档视图; 2）使用不依赖机器学习的简单启发式; 3）在没有先验知识的情况下检测零日攻击; 4）捕捉maldoc对主机系统的影响; 5）对所调查的技术进行补充，以进一步提高马尔多克攻击者的防线。

3        平台多样性

本节重点介绍为什么平台多样性可以成为检测Maldoc攻击的有效方法。我们首先给出一个激励性的例子，然后列出在发起攻击中非常重要的已识别因素，但在Windows和Mac平台上却有所不同。我们进一步展示如何使用它们来挫败攻击者，并用四个案例研究来具体化。我们最后讨论maldoc可以使用的平台检测技术，以及P LAT P AL 应该采取的预防措施。

3.1        一个激励的例子

2012年12月，研究人员发表了AAR的公开证明概念漏洞[ 37 ]。此解决方案在解析嵌入式BMP RLE编码图像（CVE-2013-2729）时会攻击PDF解析器模块中发现的堆溢出漏洞。通过简单地打开maldoc，Windows平台（包括Windows 7,8和10）上的AAR实例就会受到攻击，攻击者可以利用受感染进程的权限运行任意代码。在我们的实验中，我们在AAR 10.1.4的Windows版本上运行了这个漏洞，并重现了这个攻击。但是，当我们用AAR 10.1.4的Mac版本打开同一个样本时，攻击失败，没有发现恶意活动。

事实上，在恶意软件的历史上，Windows比Mac更吸引攻击者，maldocs也是如此。由于Windows平台的市场份额，Windows平台往往更有利可图，特别是对于大量使用和交换PDF文档的企业用户[ 38 ]。因此，期望大多数maldocs主要针对Windows平台是合理的，因为跨平台攻击由于后面讨论的因素而难以发展。

maldoc攻击者的心态和对不同平台中恶意负载的反应激励我们使用平台多样性作为maldoc检测的启发：良性文档在不同平台上打开时表现“一样，而maldoc可能有不同的”行为“在不同的平台上启动攻击。换句话说，跨平台的支持，现在可以用来使PDF格式和AAR流行起来，以防御马尔多克攻击。

3.2        多元化的因素

我们确定了与启动maldoc攻击相关的八个因素，但在Windows和Mac平台上的实施方式不同。

系统调用语义。Windows和Mac平台上的系统调用号码和用于保存系统调用参数的寄存器组都是不同的。特别是，文件，套接字，内存，进程和可执行操作都具有不重叠的系统调用语义。因此，在两个平台上构建有意义的shellcode在实践中是非常困难的。

调用约定。除了系统调用之外，用户空间函数的调用约定（即参数传递寄存器）也不同。虽然Windows平台使用rcx ，rdx 和r8 来保存前三个参数，但Mac平台使用rdi ，rsi 和rdx 。这使得类似ROP的攻击几乎不可能，因为构建这些攻击的小工具完全不同。

库依赖关系。由AAR加载的不同的库集阻止了两种类型的漏洞攻击：1）依赖加载库中的漏洞存在的漏洞，例如图形库，字体管理器或libc ，因为它们在Windows和Mac上的实现方式都不相同平台; 2）依靠加载的库中某些函数的存在的利用，例如LoadLibraryA 或dlopen 。

内存布局。从攻击点（例如，溢出缓冲区的地址或攻击者控制的整数值）到目标点的偏移，不管是返回地址，GOT / PLT条目，vtable条目还是控制数据，都不太可能跨平台是一样的。换句话说，将控制流引导到喷洒的代码通常可能被跨平台的存储器布局的差异所阻塞。

堆管理。               鉴于广泛的部署

ASLR和DEP，成功的堆缓冲区溢出通常首先导致堆元数据损坏，然后利用堆管理算法获得对控制数据（例如，vtable）的访问。但是，Windows和Mac平台之间的堆管理技术是根本不同的。因此，破坏段堆[ 67 ]（Windows分配器）维护的元数据结构的技巧在杂志malloc [ 5 ]（Mac分配器）中不起作用，反之亦然。

可执行格式。虽然Windows平台通常识别COM，NE和PE格式，但Mac平台仅识别Mach-O格式。因此，尝试在利用后加载可执行文件的maldoc将会失败。尽管可以在多个CPU架构上运行的“胖二进制文件”存在，但我们并不知道可执行格式（或任何包装器工具）能够在多个平台上运行。

文件系统语义。Windows使用反斜杠（\ ）作为路径分隔符，而Mac使用正斜杠（/ ）。另外，Windows有一个前缀盘符（例如C：\ ），而Mac有一个安装点（例如root / ）。因此，硬编码的路径名称，无论它们是在JavaScript还是由攻击者控制的shellcode，至少会在一个平台上破解。动态生成的文件名依赖于某些文件存在于给定路径的事实，这在跨平台上不太可能实现。

预期的计划/服务。这很大程度上取决于丢弃者或网络钓鱼类型的maldocs，例如，放弃使用MS Office错误的恶意MS Office文档，或将用户重定向到攻击Internet Explorer浏览器的恶意网站。由于Mac平台预计不会有这些程序，所以这些攻击在Mac平台上将会失败。

3.3        攻击分类

如图1 所示，一个典型的maldoc攻击包括三个步骤：1）发现漏洞，2）利用它们注入攻击者控制的程序逻辑，3）通过窃取信息，放弃后门，C＆C，等等。第3.2节中所确定的多样性因素可以帮助检测不同阶段的maldocs。

在查找漏洞方面，P LAT P AL 明显可以检测到特定于平台的组件上的漏洞，因为其他平台上不存在易受攻击的组件。

根据攻击是利用内存错误（例如，缓冲区溢出，整数溢出等）来劫持控制流还是利用逻辑错误（例如JavaScript API设计缺陷），开发技术可以分为两个子类。

基于内存错误的控制流劫持在开发过程中对内存内容提出了很高的要求。例如，在马尔多克样本中常见的ROP攻击需要特定的小工具和精确的信息，以便在哪里找到它们，以便进行强大的攻击。但是，这些小工具和他们在内存中的地址很容易被加载库和内存布局的差异所扭曲。

另一方面，利用自然跨平台支持的特性（例如，JavaScript隐藏的API攻击或滥用PDF文档的结构来混淆恶意负载）不受运行时内存内容的复杂性影响，并且更有可能成功。

最后，即使攻击者在前两步中成功，攻击也可以在maldoc执行恶意活动时检测到，例如执行系统调用，在Mac平台上加载PE格式的可执行文件，或访问仅存在于Windows平台。

3.4        案例研究

我们用代表性的例子来展示如何使用平台多样性来检测图1 所示的每一步中的马尔多克攻击。

平台特定的错误。平台特定错误的一个来源来自AAR使用的系统库。一个示例是CVE-2015-2426，Windows Adob​​e Type Manager Library中的整数溢出错误。详细的研究可以在[ 28 ] 找到。在这种情况下，在Windows平台上打开maldoc示例将触发利用，而在Mac平台上打开时没有任何事情发生。换句话说，利用依赖库中的错误的maldocs肯定会在其他平台上失败。

AAR实现本身也是错误的另一个来源，同时我们也发现了一些情况，在同一个平台上实现相同的功能可能容易受到攻击，而在另一个平台上却是安全的。例如，CVE-2016-4119是AAR用于解压缩嵌入图像的zlib 缩小算法中的一个免费使用漏洞[ 30 ]。AAR的Mac版本能够遍历文档并正常退出，而Windows上的AAR在渲染阶段崩溃。仔细看看它们的执行情况，可以看出解码的图像对象在这些平台上是不同的。

内存错误。由于在现代操作系统中部署了ASLR和DEP，直接的shellcode注入不能成功。因此，利用内存错误的攻击者通常需要某种形式的堆准备来获得对控制数据的读/写访问，而我们观察到的最常见的目标是vtable。

在[ 37]，maldoc示例利用CVE2013-2729（AAR本身中的整数溢出错误）来准备堆，以获得对与图像对象关联的vtable的访问。具体来说，它首先分配1000个连续的内存块，每个内存块300个字节，仔细选择一个值来匹配vtable的大小，然后在每10个块中释放一个来创建几个洞。然后，它使用300字节的畸形BMP图像来触发整数溢出错误，并设法覆盖驻留在攻击者控制的插槽（尽管攻击者不知道哪个插槽）的堆元数据。畸形BMP图像从内存中释放，但实际释放的是攻击者控制的插槽，因为堆元数据损坏。后来，

但是，这种精心构建的攻击有两个假设，不跨平台：1）Windows和Mac平台上的vtable的大小是不同的; 2）堆对象的元数据结构是不同的。因此，覆盖Mac平台上的堆元数据不会产生可观察的行为。

逻辑错误。AAR的另一个常见攻击媒介是逻辑错误，尤其是JavaScript API设计缺陷。与利用内存错误的攻击不同，JavaScript API攻击通常既不需要堆结构，也不需要ROP风格的操作。相反，它们可以用19行JavaScript代码来启动，如图2 所示。Gorenc *等*。[ 22 ]进一步扩展了这种技术，通过滥用隐藏的JavaScript API来完成远程代码执行攻击。

|  |
| --- |
| E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image011.gif  图1：在整个攻击周期中使用平台多样性来检测maldoc。× 附近的*斜体文本*是指§3.2 中可以用来检测这种攻击的因素。虚线表示在检测之后某些攻击可以存活。 |

1   var t = {};

2   t .\_\_ defineSetter \_\_（'doc' ，app.beginPriv）;

3   t .\_\_ defineSetter \_\_（'user' ，app.trustedFunction）;

4   t .\_\_ defineSetter \_\_（'settings' ，function （）{ throw 1 ;}）;

5   t .\_\_ proto\_\_ = app;

6   试试{

7   DynamicAnnotStore.call（t， null ，f）; 8 } catch （e）{}

9

10                  f（）;

11                  函数f（）{

12                  app.beginPriv（）;

13                  var file = '/c/notes/passwords.txt' ;

14                  var secret = util.stringFromStream（

15                  util.readFileIntoStream（文件，0 ）

16                  ）;

17                  app.alert（秘密）;

18                  app.endPriv（）;

19                  }

图2：CVE-2014-0521概念证明的利用

除了构建简单之外，由于JavaScript的跨平台支持，这些攻击通常在Windows和Mac平台上均可用。因此，通过平台多样性来检测这些攻击的关键是利用文件系统语义，预期安装程序等不同系统组件的不同，并在执行恶意活动时寻找执行差异。例如，在图2的示例中，第15行将在Mac平台上失败，因为Mac上不存在这样的文件路径。

3.5        平台感知的利用

鉴于使用相同有效载荷在不同平台上启动马尔多克攻击的难度，攻击者可以做的是首先通过显式或隐式通道检测maldoc运行在哪个平台，然后利用平台特定的有效载荷发起攻击。

特别是，Adobe JavaScript API包含可公开访问的函数和对象字段，可以在不同平台上执行时返回不同的值。例如，app.platform 在各自的平台上返回WIN 和MAC 。Doc.path 返回打开文档的文件路径，通过测试返回的路径是否以/ c / 为前缀来检查文档是否在Windows或Mac上打开。

启动平台感知攻击的另一种方法是将攻击嵌入到两个平台特定的漏洞中，每个漏洞都针对一个平台。这样，无论在哪个平台上打开maldoc，都会触发一个漏洞，并可能发生恶意活动。

事实上，尽管在我们的样本集中，平台意识的maldocs很少，但是P LAT P AL 必须意识到这些攻击方法，并且采取预防措施来检测它们。特别是攻击者在启动攻击之前可以先探测平台的可能性意味着仅仅比较外部行为（例如，文件系统操作或网络活动）可能是不够的，因为相同的外部行为可能是由于不同的攻击。在不追踪内部PDF处理的情况下，maldocs可以很容易地避开P LAT P AL 的检测，例如，通过携带多个ROP有效载荷，并根据返回值动态决定使用哪个有效载荷app.platform ，甚至使用JIT-ROP [ 49 ] 等技术动态生成ROP负载。

但是，我们确实承认，鉴于PDF规范的复杂性，P LAT P AL 没有列举所有可能的平台探测技术。因此，P LAT P AL 可能会通过我们尚未发现的隐性渠道（例如，定时副通道）被回避。

3.6        平台不可知的利用

我们还确定了一些可以帮助“中和”由平台多样性引起的不确定性的技术，包括但不限于堆风水，堆喷和多边形shellcode。

堆风水。通过使连续的堆分配和精心挑选的尺寸的去分配，攻击者可以系统地操纵堆的布局和预测下一分配或解除分配[地址51 ]。即使不知道系统内存分配器的每个细节，这也增加了访问关键数据（如vtables）的机会。

堆喷和NOP雪橇。通过在堆中重复分配攻击有效载荷和NOP底座[ 13 ]，攻击者从使用精确的存储位置缓解控制流劫持; 相反，攻击者只需要确保控制流程被重定向到喷射区域。

Ployglot shellcode蹦床。虽然在野生没有看到，有可能构建操作系统无关的shellcode以相似的方式作为CPU架构无关的shellcode [ 17 ，64 ]。关键的想法是在一个平台上找到有意义的操作，在另一个平台上找到有意义的操作，并使用这些操作跳转到不同的平台特定活动的代码。

尽管这些操作在两个平台上都能成功，但使用这些技术的攻击仍然可以通过平台多样性来检测。这是因为这些操作必须与其他程序配对才能完成端到端的攻击。例如，堆操作可以成功，但由于堆管理的差异，堆结构的内存布局可能不适合两种平台将关键数据放置在攻击者控制的内存中，而ployglot shellcode蹦床可以在不破坏AAR的情况下运行，但攻击仍然可以被执行的恶意活动检测到。

4        所述p LAT P AL 方法

本节介绍了设计P LAT P AL 所面临的挑战及其解决方案，可为Maldoc检测获得平台多样性。

4.1        双层追踪

虽然平台多样性的启发式听起来很直观，但是还是会出现两个自然的问题：1）不同平台之间的“行为”可能会有什么不同？2）他们如何被普遍追踪和比较？

要回答第一个问题，“行为”必须满足两个要求：1）它们是可用的，不跨平台变化; 2）对于良性文件，它们是一样的，对于多元文化可能不同。为此，我们确定了两套符合以下要求的“行为”：AAR的内部PDF处理功能（*内部行为*）以及执行文档（*外部行为*）时对主机系统的外部影响。

对于*内部行为*，在AAR中，PDF文档按确定的顺序通过PDF处理函数，并按顺序触发预定义的回调。例如，当对象被类似或呈现时，发出回调。当比较跨平台的执行时，对于良性文档，由于AAR的跨平台支持，函数执行顺序和结果是相同的，而对于maldoc，执行跟踪在许多地方可能不同，取决于攻击的方式完成了。

在*外部行为方面*，由于PDF规范的跨平台性，如果一些合法的行为在一个平台上影响到主机系统，预计在另一个平台上打开文档时会显示同样的行为。例如，如果良性文档连接到远程主机（例如，用于内容下载或表单提交），则其他平台上的行为也是相同的。但是，如果互联网连接仅在成功利用后触发，则不会在失败的平台上显示。

P的架构LAT P AL 中描述图3 。P LAT P AL 追踪内部和外部行为，我们认为追踪这两种类型的行为是必要的。追踪外部行为对于捕捉成功利用后的行为差异（即图1中的恶意行为步骤）至关重要。例如，在一个成功的JavaScript隐藏API攻击[ 22 ]之后，攻击者可能想要执行shellcode，由于系统调用语义的差异，在Mac上将会失败。但是，内部行为都显示相同的事情：JavaScript代码的执行停在相同的地方。

最有说服力的理由有一个内部行为示踪剂是打败平台探测的尝试，没有其中p LAT P AL 可以通过启动平台感知的攻击很容易回避，如在3.5节。跟踪内部行为的另一个原因是提供一些有关哪个AAR组件被利用或发生攻击的信息，这有助于分析maldoc样本，尤其是概念验证（PoC）样本，这些样本只是在没有任何外部活动的情况下使AAR崩溃。

4.2        内部PDF处理

P LAT P AL 的内部行为跟踪程序密切关注AAR如何处理PDF文档。AAR内部的PDF处理可以分为两个阶段。

在*解析*阶段，打开底层文档并扫描标题以快速定位预告片和交叉引用表（XRT）。在找到XRT之后，枚举和分析称为COS对象的PDF文档的基本元素。请注意，COS对象只是具有类型标签的数据（例如，整数，字符串，关键字，数组，字典或流）。然后根据AAR对PDF规范的解释，将一个或多个COS对象组装成PDF特定的组件，如文本，图像，字体，表单，页面，JavaScript代码等。PDF文档的层次结构（例如，哪些文本出现在特定的页面中）也是沿着这个过程构建的。输出，称为PD树，然后传递给渲染引擎进行显示。

|  |
| --- |
| E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image013.gif  图3：P LAT P AL 架构 将可疑文件提交给两台具有不同平台的虚拟机。在执行过程中，内部和外部行为都被追踪和比较。任何行为的分歧都被视为恶意信号。 |

如果有的话，*绘图*阶段首先执行由文档指定的OpenAction 。几乎所有的maldocs都会注册任何可能在OpenAction中触发其恶意负载的东西，以便在文档打开时立即利用。随后的绘画活动取决于用户的输入，例如向下滚动到下一页触发该页的呈现。因此，在这个阶段，P LAT P AL 不仅钩住了功能，而且还主动地驱动文档渲染组件。请注意，显示内容到屏幕是一个平台相关的程序，因此，不会被P LAT P AL 挂钩，但回调（例如，一个对象呈现）是平台无关的，将被追踪。

另外，对于AAR，当呈现引擎执行JavaScript动作或者绘制JavaScript嵌入形式时，将执行整个JavaScript代码块。但是，这也使得§3.5中描述的平台检测尝试成为可能，并且容易逃脱P LAT P AL 的检测。为了避免这种情况，P LAT P AL 旨在抑制JavaScript代码的自动块执行。相反，代码被标记为一系列逐个执行的语句，并记录每个执行的结果并随后进行比较。如果语句调用用户定义的函数，那么该函数也是逐步执行的。

以下是每个步骤中记录的跟踪摘要：COS对象解析：P LAT P AL 输出COS对象（包括类型和内容）的解析结果。

PD树构造：P LAT P AL 输出PD树中每个具有类型和分层位置的PD组件。脚本执行：P LAT P AL 输出每个执行的语句和相应的结果。其他操作：P LAT P AL 输出执行此操作期间触发的每个回调，例如更改页面查看或访问的URL。

元素渲染：P LAT P AL 输出在渲染PDF元素期间触发的每个回调。

4.3        外部系统影响

由于系统调用是程序与主机平台交互的主要机制，P LAT P AL 挂接系统调用并记录参数和返回值，以便捕获在主机系统上执行maldoc的影响。但是，对于P LAT P AL ，在比较跨平台的系统调用时会出现一个独特的问题，因为Windows和Mac上的系统调用语义有很大不同。

为了简化跨平台外部行为的比较，P LAT P AL 从原始系统调用转储中提取高级活动。P LAT P AL 特别关注三类活动：

文件系统操作：包括在执行文档期间打开/创建的文件，以及文件删除，重命名，链接等。网络活动：包括远程套接字的域，IP地址和端口。

外部可执行文件启动：包括在打开文档后执行任何程序。

除了从系统调用轨迹构建的行为之外，

P LAT P AL 附加监控文件的打开期间AAR是否正常或崩溃退出。我们（经验）认为，在这些高级行为抽象中，可以捕获许多典型的恶意软件活动，如窃取信息，C＆C，落后等。这种做法也对准与许多自动化恶意软件分析工具，如杜鹃[ 44 ]和了CWSandbox [ 63 ]，它也自动生成排序和组织恶意软件的行为成几个类别的摘要。然而，不像这些动态的恶意软件分析工具，它们根据这些活动的顺序或层级来推断样本的恶意程度，这是对P LAT PAL 是所捕获的一组活动在不同平台上有所不同。另一个区别是，由Cuckoo和CWSandbox生成的摘要通常需要手动解释来判断恶意，而来自P LAT P AL 的摘要不需要人工比较跨平台的行为。

5        实施

P LAT P AL 由三部分组成：1）AAR插件形式的内部行为示踪器; 2）syscall示踪器形式的外部行为示踪器; 3）基于VMware的动态文档审查的沙箱环境。我们将P LAT P AL 原型设计为在最近的Windows（版本7,8和10）和Mac（版本Yosemite，El Capitan和Sierra）平台上工作，并且与从Adobe Reader X 10.0到最新版本的所有AAR版本兼容Adobe Acrobat DC。

5.1        内部行为示踪

鉴于AAR是封闭源码软件，通过源代码工具来钩住AAR的PDF处理功能是不可行的。最初，我们使用动态二进制仪器工具（即，英特尔引脚[ 32 ]和DynamoRio [ 7]）钩住AAR的执行，并在运行时检查函数调用。然而，这种方法有两个显着的缺点：1）这些工具引入了16-20倍的减速，这对于实际的马尔多克检测是不可容忍的。例如，执行一个两页的文档可能需要五分钟的时间，有时甚至会被系统暂停; 2）PDF处理逻辑隐藏在超过15000个功能（最新版本的AAR）中，没有名称或符号信息。即使不是不可能确定关键功能，也很难构建整个周期。

为此，P LAT P AL 选择开发一个AAR插件作为内部行为追踪器。AAR插件技术[ 3 ]旨在扩展AAR的功能，如数据库交互，在线协作等。编译后的插件采用Windows上可加载DLL的形式，以及由AAR加载的Mac上的应用程序包初始化并在运行时对AAR进行有效的控制。AAR插件提供了一些适合P LAT P AL的很好的功能1）跨平台支持将平台特定的操作抽象到更高层次; 2）它采用AAR的内部逻辑在PDF处理和总结了逻辑到782层的功能和回调（很好的分类和标记的），这使得P LAT P AL 两者被动地监视这些功能的执行，并积极驱动的文档，其中包括执行JavaScript代码和渲染元素; 3）在AAR版本中是稳定的（从版本10开始只增加了两个函数，P LAT P AL 不使用）; 4）由于AAR插件是可加载模块的形式，因此它将总文档分析时间缩短为平均24秒。

在§4.2中讨论的记录行为中，COS对象和PD分层信息是使用提供的枚举方法提取的

CosDoc ，PDDoc 和PDF\_Consultant 类。JavaScript代码首先由词法分析器进行标记

SpiderMonkey，并从AcroForm 类的AFExecuteThisScript 方法执行语句的声明。其余的PDF支持的动作是使用AVDocPerformAction方法启动的。AAR插件中的PDF处理函数可以通过简单的JMP-Trampoline热补丁技术挂钩，如[ 6 ]中所述。

5.2        外部行为示踪

如§4.3所示，P LAT P AL 的外部行为跟踪器在文档执行期间记录系统调用参数和返回值。在Windows上，跟踪器是基于NtTrace [ 41 ]实现的; 在Mac上，示踪剂是利用BSD系统上可用的DTrace [ 9 ]机制的Dscript 。这两种技术都在各自的平台成熟和招致小执行开销：相比于发射AAR没有附加示踪剂，这有助于控制每个文件的总执行时间的15％至35％。构建高层次的行为与Cuckoo客座[ 44 ] 类似。

在P LAT P AL中，系统调用追踪只有在AAR打开文档后才会启动。AAR初始化过程没有被跟踪（因为AAR本身不是恶意软件）并且P LAT P AL 在启动过程中没有混乱的文件系统活动（例如，加载库，初始化目录结构等），留下执行痕迹在一个非常短暂和干净的状态。实际上，良性文件通常有大约20个文件系统跟踪条目，没有网络活动或外部程序启动。AAR使用单个线程来加载和解析文档，并在文档呈现期间生成一个助手线程。两个线程的系统调用都被跟踪和比较。

为了比较文件路径，P LAT P AL 进一步聚合并标记文件系统操作跟踪到几个在两个平台上都有映射的类别，包括AAR程序逻辑，AAR支持文件，AAR工作缓存，系统库/框架依赖关系，系统字体和临时文件。这些标签以外的文件将转到未知类别，并将根据文件名进行比较。

5.3        自动执行沙箱

对于P LAT P AL ，具有自动执行沙箱的目的是双重的：1），以限制一个适当的边界内的恶意活动和2），以提供针对每个文件检验一个干净的执行环境，从副作用先前执行免费。

使用clean-slate操作系统初始化虚拟机（VM），并随后调配必要的工具和设置，包括AAR，插件和系统调用示踪程序。内存和磁盘快照在提供之后进行，每个后续文档执行都会从此快照中恢复状态。P LAT P AL 使用VMware的虚拟机和快照的管理。

流程。                      P LAT P AL       可以    被    启动      像

PlatPal <文件检查> 。之后，P LAT P AL 填充Windows VM和Mac VM，并恢复内存和磁盘快照。然后将可疑文档上传到这些虚拟机，并且AAR以附加的系统调用跟踪器启动。AAR完成初始化后，控制权被转移到插件（内部追踪器），打开文档进行检查。考试结束后（或AAR崩溃），内部和外部追踪记录将从相应的虚拟机中提取出来，并在主机上进行比较。P LAT P AL 报告是否在这些日志中检测到差异。

6        评估

在本节中，我们验证了P LAT P AL 的基本假设：良性文档在不同平台上打开时表现相同，而在不同平台上进行开发时，maldoc表现出不同的表现。我们还根据完成分析所花费的总时间来评估P LAT P AL 的表现。

实验设置。实验在MacBook Pro（2016款）上进行，搭载Intel酷睿i7 2.9GHz CPU和16GB RAM，运行macOS Sierra。一个VM配置了Windows 7 Professional SP1，另一个VM配置了OSX Yosemite 10.10.1。每个虚拟机进一步配置了6个不同版本的AAR实例[[1]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftn1" \o ")，如表2 所列。每个文档样本在执行一分钟后被强制关闭。

6.1        良性样品

良性样本集由三部分组成：1000个样本通过搜索Google搜索文件类型PDF和没有关键字。但是，大多数这些样本不使用通常由maldocs利用的功能。例如，只有28个文件包含嵌入的字体，6个文件包含JavaScript代码。因此，我们进一步收集了来自PDF学习网站[[2]的](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftn2" \o ") 30个使用PDF标准的高级功能的样本，包括嵌入式JavaScript（26个样本），AcroForm（17），自定义字体

（6）和3D对象（2）。所有的样本都提交给VirusTotal并由48个AV产品进行扫描，并且任何AV引擎都不会将其标记为恶意。

样本提交给P LAT P AL 进行分析。特别是，每个文档都在两个平台上由所有六个AAR实例版本打开。这是为了验证所有的AAR阅读器实例在文档执行过程中不引入不确定性。每个AAR版本进行成对行为比较，对于任何测试的AAR版本，没有观察到差异。更重要的是，实验结果支持P LAT P AL 假设的第一部分：良性文档在不同平台上表现相同。

6.2        马尔多克检测

maldoc样本全部从VirusTotal收集。具体来说，我们收集了截至2016年12月31日的具有确定的CVE编号的样本（即样本利用特定的CVE）[[3]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftn3" \o ")。作为原型，我们通过分析2013年后发布的CVE来限制范围，并进一步过滤样本明显是错误标记的（例如，利用2016 CVE的2011年样本）或错误类型（例如zip文件或图像文件），并获得320个样本的数据集。

样本提交给P LAT P AL 进行分析。另外，我们根据CVE发布的时间选择最受欢迎的AAR版本。换句话说，每个漏洞攻击都是零日攻击

AAR版本测试。每个CVE检测结果在表2 中给出，表3中列出了哪些行为因素导致差异的细分。

解释。对于提交给P LAT P AL的任何样本，只有三个结果是可能的：

1）  *恶意*：至少有一个行为差异，包括两个平台上的AAR崩溃但内部行为不同的情况，即它们在不同的PDF处理阶段崩溃。

2）  *可疑*：两个平台的AAR崩溃，但内部行为没有差异。鉴于良性文件没有理由使AAR崩溃，P LAT P AL 认为这些样本是可疑的。

3）  *良性*：没有行为差异可以观察到，并且AAR在两个平台上都优雅地退出。

总体结果。在320个样本中，P LAT P AL 检测到*恶意*样本209个（65.3％），*可疑*样本34个（10.6％），*良性*样本77个（24.1％）。

可疑的样品。在34个可疑样本中，我们能够确认16个是PoC样本，包括Exploit-DB [ 19 ] 上发布的7 个，公共博客中的3个，以及由VirusTotal记录的原始文件名推断的6个样本。

这些样本很可能是通过模糊的方式获得的

                            AAR            Num。                     结果

CVE                                                         E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image014.gif

                         版本       示例     两个崩溃        分歧

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2016-6946 | DC.16.45 | 51 | 8 | 40 |
| 2016-4204 | DC.16.45 | 78 | 7 | 37 |
| 2016-4119 | DC.10.60 | 1 | 0 | 1 |
| 2016至1091年 | DC.10.60 | 63 | 6 | 31 |
| 2016年至1077年 | DC.10.60 | 1 | 0 | 1 |
| 2016年至1046年 | DC.10.60 | 4 | 0 | 4 |
| 2015-5097 | 11.0.10 | 4 | 0 | 4 |
| 2015年至2426年 | 11.0.10 | 14 | 6 | 8 |
| 2015-0090 | 11.0.10 | 1 | 0 | 1 |
| 2014-0521 | 11.0.00 | 2 | 0 | 2 |
| 2014-0495 | 11.0.00 | 2 | 0 | 2 |
| 2013-3353 | 10.1.4 | 16 | 4 | 10 |
| 2013-3346 | 10.1.4 | 7 | 0 | 7 |
| 2713至29年 | 10.1.4 | 23 | 3 | 19 |
| 2013-0640 | 10.1.0 | 三十 | 0 | 22 |
| 2013-0641 | 10.1.0 | 23 | 0 | 20 |
| 总 |  | 320 | 34 | 209 |

表2：根据CVE编号分组的P LAT P AL maldoc检测结果。*两次崩溃都*意味着AAR在执行maldoc样本时在两个平台上崩溃，内部行为没有分歧; *分歧*意味着至少有一个行为差异（内部或外部）被观察到。

执行，只会使AAR崩溃。我们预计它也适用于其他可疑样本。

良性样本。我们确定了这些样本检测失败的几个原因。

1） maldoc的目标是老的和特定的AAR版本。虽然大多数maldoc样本利用了各种各样的AAR版本，但我们确实找到仅针对旧的AAR版本的样本，即9.X和8.X，其中包括8个CVE-2013-0640样本，3个CVE-2013-0641样本和1个CVE-2013-2729样本。我们还发现，13个CVE-2016-4204样本和10个CVE-2016-1091样本似乎正在利用AAR 11.0.X版本，并且该漏洞在实验中使用的AAR DC版本上不起作用。这是基于手动检查这些示例中的JavaScript转储。

在77个归类为良性的样本中，总共有36个。这也说明了动态分析方法P LAT P AL 的缺点，它需要适当地设置执行环境来诱发恶意行为。

2） VirusTotal上的AV供应商可能会错误分类maldoc样本。这对于11个CVE-

2016-4204和8个CVE-2016-1091样本，因为在VirusTotal上托管的48个AV产品中，只有一个AV供应商将其标记为恶意。总的来说，这在77个分类为良性的样品中占了19个。

3）

|  |
| --- |
|  |
|  | E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image015.gif |

maldoc不会执行恶意活动。并非maldoc中的所有恶意活动都可以触发。特别是，我们观察到两个CVE-2013-3353样本

试图用JavaScript连接到C＆C服务器，但之后没有做任何反应，

这导致执行轨迹中没有分歧。

最后，其余样本分为良性样本（共20个样本），我们无法确定没有观察到行为差异的原因。这可能是由于任何上述原因（但我们无法证实），我们不排除某些样品可能逃避P中的可能性LAT P AL 的检测。鉴于PDF说明书的范围和灵活性，能够将P LAT P AL 需要钩住多个功能（例如，每字形到主机编码转换执行的字体）来捕获更细粒度的内部行为。

行为有效性。表3 还显示了检测maldocs的各种行为的有效性。

1）  第一行，可能只有外部行为差异，而内部行为是相同的（例如，由于纯粹的JavaScript攻击）。在第一列中，也可能只有内部行为分歧，而外部行为是相同的（由于AAR的强大的纠错能力）。

2）  崩溃/不崩溃是最有效的外部指标，因为内存错误利用是样本中maldoc攻击的主要技术。JavaScript执行是最有效的内部指标，因为几乎所有的攻击都涉及JavaScript; 甚至内存错误利用它来准备堆。

通过内部追踪来定位攻击。内部跟踪器的一个补充目标是提供洞察哪个AAR组件被利用或攻击发生在给定的maldoc样本。为了评估这个目标是如何实现的，我们对内部行为发散的位置和每个CVE的目标AAR组件进行了交叉检查[[4]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftn4" \o ")。结果如表4 所示。在七宗个案中，有四宗是P LAT P AL内部追踪者在调用易受攻击的组件时发现有分歧。在CVE-2015-2426的情况下，由于易受攻击的组件是字体库，所以在渲染过程中首先检测到发散。在CVE-2013-3346的情况下，通过JavaScript代码触发易受攻击的组件（ToolButton回调），因此脚本引擎中会出现第一个分歧。在CVE-2013-2729的情况下，尽管bug在解析器组件中，但是当maldoc正在堆放风水来排列堆对象时，检测到发散。

抵御自动化maldoc世代。我们使用先进的maldoc生成工具EvadeML [ 65 ] 测试P LAT P AL 的恢复能力，EvadeML [ 65 ]根据给定的恶意种子文件自动产生表1 中ML取决于方法的回避maldoc变种。为此，我们从209个恶意样本中选择了30个样本，这些样本也被PDFrate [ 46 ] 检测为恶意样本，

|  |  |  |  |
| --- | --- | --- | --- |
| CVE | 目标组件 | 分歧首先发生 | 检测 |
| 2016-4119 | 分析器 | 分析器 | Vuln。零件 |
| 2016年至1077年 | 分析器 | 分析器 | Vuln。零件 |
| 2016年至1046年 | 脚本引擎 | 脚本引擎 | Vuln。零件 |
| 2015年至2426年 | 图书馆 | 给予 | 利用载体 |
| 2014-0521 | 脚本引擎 | 脚本引擎 | Vuln。零件 |
| 2013-3346 | 给予 | 脚本引擎 | 利用载体 |
| 2713至29年 | 分析器 | 脚本引擎 | 利用载体 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image016.gif   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | 没有不同 | 这两个崩溃 | 一次崩溃 | 文件系统 | 网络 | 可执行文件 |  | | 没有不同 | 77 | 34 | 0 | 6 | 3 | 0 | 120 | | COS对象解析 | 4 | 8 | 23 | 0 | 0 | 0 | 35 | | PD树建设 | 0 | 0 | 2 | 4 | 2 | 0 | 8 | | JavaScript执行 | 五 | 五 | 47 | 18 | 12 | 4 | 91 | | 其他行为 | 0 | 0 | 0 | 2 | 0 | 2 | 4 | | 元素渲染 | 3 | 10 | 35 | 9 | 五 | 0 | 62 | | 总 | 89 | 57 | 107 | 39 | 22 | 6 | 320 |   表3：P LAT P AL 通过使发散的因素分组maldoc的检测结果。请注意，对于每个样本，只有一个内部因素和一个外部因素被视为发散的原因。例如，如果一个示例在Mac上崩溃，并且在Windows上不崩溃，即使它们的文件系统活动不同，它也会记录在崩溃/无崩溃类别中。同样的规则适用于内部行为。  E:\bdfirst\2018\2018Q1\sec17-xu-meng.files\image017.gif |

表4：由P LAT P AL 的内部跟踪器检测到的发散与实际的越野车AAR组件相比较。

与EvadeML [[5]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftn5" \o ")一起使用的默认PDF分类器。然后，我们使用EvadeML来突变这些样本，直到所有变体被认为是良性的。最后，我们将这些回避变量发送给P LAT P AL 进行分析，并将其全部再次标记为恶意行为，即仍然观察到行为差异。这个实验验证了P LAT P AL 在自动化maldoc生成工具上的弹性。弹性化的主要原因是EvadeML主要关注于改变maldoc的结构特征，同时保留其开发逻辑以及发起攻击时的内部和外部行为。

6.3        性能

P中LAT P AL ，总分析时间由两个部分组成：1）时间来恢复磁盘和存储器快照和2）的时间来执行文档样本。后者可以进一步分解为文档解析，脚本执行和元素渲染时间。表5 显示了每个项目的时间和整个执行时间。

平均而言，两台虚拟机上的文件执行速度可以几乎同时完成（23.7 vs 22.1秒）。鉴于虚拟机可以并行运行，完整的分析可以在25秒内完成。一个显着的差异

                                             大道       标准。     大道      标准。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 快照还原 | 9.7 | 1.1 | 12.6 | 1.1 |
| 文件解析 | 0.5 | 0.2 | 0.6 | 0.2 |
| 脚本执行 | 10.5 | 13.0 | 5.1 | 3.3 |
| 元素渲染 | 7.3 | 8.9 | 6.2 | 6 |
| 总 | 23.7 | 8.5 | 22.1 | 6.3 |

表5：p个击穿LAT P AL （：秒单位）每文档的分析时间。

在Windows平台上的脚本执行比在Mac平台上花费的时间要长得多。这是因为几乎所有的maldoc样本都是针对Windows平台的，并使用JavaScript来发起攻击。这种攻击很快就会在Mac上失败（例如，ROP小工具的地址错误）并使AAR崩溃，但在Windows上成功，因此需要花费更长时间才能完成。同样的原因也解释了为什么在Windows平台上脚本执行时间的标准偏差较大。

7        讨论

7.1        限制

用户交互驱动的攻击。虽然P LAT P AL 能够模拟简单用户的交互（例如，滚动，按钮点击等），但P LAT P AL 并不试图探索所有可能的行为（例如按键，表格填充等）或探索所有JavaScript代码的分支。同样，P LAT P AL 也不能检测到故意延迟执行的攻击（例如，在打开文档两分钟后开始利用）。这些是任何动态分析工具的常见限制。但是，我们认为这不是马尔多克检测的严重问题，因为隐藏

复杂的用户交互之后的恶意活动限制了其牺牲受害者系统的有效性。

社会工程攻击。P LAT P AL 无法检测旨在执行社交工程攻击的maldoc，例如使用JavaScript窗口伪造密码提示或诱使用户下载文件并执行它。这是因为这些maldocs既不利用AAR中的漏洞，也不注入恶意负载（实际上它们是结构上合法的文档），因此在两个平台上都具有完全相同的行为。

有针对性的AAR版本。如果maldoc目标AAR的特定版本，其P中的行为LAT P AL 将可能是要么撞毁两个AAR情况下（即，所利用的漏洞，但使用了错误的有效载荷），或文档呈现优雅地关闭，因为纠错由AAR。在后一种情况下，P LAT P AL 将无法检测到行为差异。这通常不是对于P问题LAT P AL 在实践中，为P LAT P AL 将主要用于检测抗AAR的最新版本maldocs。但是，P LAT P AL也可以在许多AAR版本上对文档进行测试，并且只要在任何单一版本中观察到差异就将其标记为可疑。

非确定性。                另一个潜在的问题

P LAT P AL 是文件执行中的非确定性因素可能导致错误警报。示例包括gettime 函数的返回值或通过JavaScript代码提供的随机数生成器。虽然P LAT P AL 在实验过程中没有遇到这样的问题，但是一个完整的解决方案需要对PDF JavaScript规范进行彻底的检查，并确定所有的非确定性。这些非确定性因素需要在一个平台上执行文档时记录下来，并在另一个平台上重播。

7.2        部署

由于P LAT P AL 需要至少两个虚拟机，因此需要大量的图像和内存来支持P LAT P AL 的操作。我们目前的实施使用

60GB磁盘空间来承载六个AAR版本的快照和每个正在运行的VM的2GB内存。

为此，我们认为P LAT P AL 最适合于可以使用P LAT P AL 定期扫描现有文件或新上传文件的云存储提供商（例如Dropbox，Google Docs，Facebook等）。这些提供商可以承担建立具有不同平台的虚拟机所需的磁盘和内存，并享受规模经济。同样，P LAT P AL 也适合VirusTotal等在线恶意软件扫描服务或防病毒产品的云版本。

另外，作为补充方案，P LAT P AL 可以很容易地与之前的作品（表1 ）整合，以提高其检测精度。具体而言，P LAT P AL 的内部行为跟踪器可以用来替代这些技术中的解析器来缓解解析器混淆攻击[ 11 ]。COS对象和PD树信息可以被馈送到基于元数据的技术[ 33 ，36 ，46 ，52 ]，而JavaScript代码转储可以被馈送到JavaScript的定向技术[ 14 ，27 ，31 ，45 ，48 ，58 ，59 ]进行分析。

7.3        未来的作品

我们相信P LAT P AL 是一个灵活的框架，不仅适用于基于PDF的maldoc检测，而且适用于系统地处理安全性和多样性。

支持更多的文档类型。MS Office程序与AAR产品共享许多功能，例如1）支持Windows和Mac平台; 2）支持插件架构，允许有效地挂接文件处理功能和行动驾驶; 3）基于由静态组件（例如文本）和可编程组件（例如，宏）组成的标准规范来执行文档。因此，我们并不认为将P LAT P AL 移植到支持针对MS Office套件的Maldoc检测方面存在根本性的困难。

另外一个例子，由于网站也可以被视为嵌入式JavaScript的HTML文档，恶意网站检测也适用于P LAT P AL 的框架。此外，由于Chrome和Firefox浏览器及其脚本引擎是开源的，因此P LAT P AL 能够执行更细粒度的行为跟踪，并与源代码工具进行比较。

探索建筑多样性。除了平台多样性以外，CPU架构的多样性也可以用于maldoc检测，我们预计在阻止maldoc攻击方面也会有类似的效果。为了验证这一点，我们计划扩展P LAT P AL 以支持Android版本的AAR，它同时具有ARM和x86两种版本。

8        额外的相关工作

除了maldoc检测工作，作为一个Nversion系统，P LAT P AL 也与N版本的研究有关。N版本系统的概念最初是作为一种软件容错技术引入的[ 12 ]，后来被用于增强系统和软件的安全性。例如，Frost [ 60 ]用补充调度算法来处理并发错误; 起重机*等*。[ 16 ]应用动态控制流分集和噪声注入来阻止高速缓存侧信道攻击; Tightlip [ 68 ]和Capizzi *等*。[ 10]将程序变体中的敏感数据随机化，以减轻隐私泄漏; Mx [ 24 ]使用相同程序的多个版本来存活更新错误; 鸡尾酒[ 66 ]使用多个网页浏览器实施，以生存供应商特定的攻击; 和Nvariant [ 15 ]，Replicae [ 8 ]和GHUMVEE [ 61 ]在不相交的内存布局中运行程序变体，以缓解代码重用攻击。同样，Orchestra [ 43 ]同步两个程序变体，它们在相反的方向上增长堆栈以进行入侵检测。特别是，Smutz *等*。[ 47]通过构建多样化的分类器，将它们集成为一个单一的系统，并将它们的分类输出与相互协议分析进行比较，从而识别和防止检测漏洞。

虽然P LAT P AL 是为了一个完全不同的目标而设计的（即，maldoc检测），但它与N版本系统分享了洞见：攻击者不得不同时用相同的输入来折衷所有变体，以便取消或误导整个系统。

另一个相关的工作是向执行环境引入多样性，以吸引和检测恶意行为。例如，HoneyClient [ 56 ]从网络流（例如PDF文件）缓存和类似潜在的恶意对象，然后将其发送到多个模拟环境进行分析。Balzarotti *等*。[ 4 ]通过比较跨运行时的运行时行为，检测恶意软件中的“分裂人格”，即在模拟环境和裸机中显示分歧行为的恶意软件。Rozzle [ 26 ]使用符号执行来模拟恶意软件通常会检查的不同环境值，因此可以从恶意软件中吸引特定于环境的行为。显示分歧的行为。

P LAT P AL 与这些作品有相同的信念：多样化的执行环境导致了多样化的行为，并且侧重于收集平台多样性以进行马尔克洛克检测。

9        结论

由于AAR的持续开发，maldoc检测已经成为一个紧迫的问题。对现有技术的调查显示，他们很容易受到最近的攻击，如解析器混淆和ML规避攻击。针对这一点，我们提出了一个新的视角：平台多样性，以及用于maldoc检测的原型P LAT P AL。P LAT P AL挂钩到AAR以跟踪内部PDF处理，还使用全面的动态分析来捕获maldoc对主机系统的外部影响。内部和外部的痕迹都进行了比较，唯一的启发式检测maldoc是基于这样的观察：良性文档在不同平台上表现相同，而maldoc在开发过程中表现不同，因为系统调用，内存管理等的多样化实现跨平台。这样的启发式算法不需要已知的maldoc样本来导出区分maldoc和良性文档的模式，这也使得P LAT P AL 能够在事先不知道攻击的情况下检测到零日攻击。评估显示P LAT P AL在良性样本中不会产生虚假警报，在恶意样本中检测到各种行为差异，是一个可扩展和实用的解决方案。

10        致谢

我们感谢我们的牧羊人Alexandros Kapravelos和匿名审稿人的有益反馈。这项研究得到了NSF的支持，获得DGE-

1500084，CNS-1563848，CRI-1629851，CNS-1017265，

美国空军根据合同号FA8650-10-C-7025，DARPA根据合同编号CNS-0831300和CNS-1149051，ONR根据合同号N000140911042和N000141512162，DHS根据合同号N66001-12-C-0133， DARPA FA8650-15-C-7556和DARPA HR0011-16-C-0059，以及授予MSIP / IITP [B0101-15-0644]的ETRI。

参考

[1]      Adobe Systems Inc.的文档管理-可移植文档格式，2008. [http://wwwimages.adobe.com/content/dam/ 的Adobe / EN / DEVNET / PDF / PDF文件/ PDF32000\_2008.pdf 。](http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf)

[2]      Adobe Systems Inc. Adobe Reader简介受保护

Mode，2010. [http://blogs.adobe.com/security/2010/ 07 / introduction-adobe-reader-protected-mode.html 。](http://blogs.adobe.com/security/2010/07/introducing-adobe-reader-protected-mode.html)

[3]      的Adobe 系统       公司    插件       和    应用，        2015年 [http://help.adobe.com/en\_US/acrobat/acrobat\_dc\_ SDK / 2015 / HTML帮助/＃吨= Acro12\_MasterBook / Plugins\_ 介绍/ About\_plug-ins.htm 。](http://help.adobe.com/en_US/acrobat/acrobat_dc_sdk/2015/HTMLHelp/#t=Acro12_MasterBook/Plugins_Introduction/About_plug-ins.htm)

[4]      大卫巴扎罗蒂，马可·科瓦，克里斯托夫·卡尔伯格，克里斯托弗·克鲁格尔，恩金·科尔达和乔凡·维尼亚。有效的检测

在恶意软件中分裂人物。在*第十七届年会会刊上*

*网络和分布式系统安全研讨会（NDSS）*，加利福尼亚州圣迭戈，2010年2月 - 3月。

[5]      泰勒Bohan。在区域：OS X堆开发。在*2016年Summercon论文集*，新Yark，NY，2016年7月。

[6]      Jurriaan Bremer。86 API挂钩揭秘，2012年[的https：//jbremer.org/x86-api-hooking-demystified/ 。](https://jbremer.org/x86-api-hooking-demystified/)

[7]      德里克·布鲁宁。*高效，透明，全面的运行时代码操作*。博士论文，麻省理工学院，2004年。

[8]      达尼洛·布鲁斯基，洛伦佐·卡瓦拉罗和安德里亚·兰齐。用于击败内存错误漏洞的多元化进程副本。在*2007年的国际行为，计算机和通信大会（IPCCC）论文集*，新奥尔良，洛杉矶，2007年4月。

[9]      Bryan M. Cantrill，Michael W. Shapiro和Adam H. Leventhal。生产系统的动态仪表。在*2004年的USENIX年度技术会议（ATC）的论文集*，马萨诸塞州波士顿，2004年六月至七月。

[10]   Roberto Capizzi，Antonio Longo，VN Venkatakrishnan和A. Prasad Sistla。通过暗影执行防止信息泄漏。在*软件工程2008年国际大会（ICSE）论文集*，阿纳海姆，CA，2008年12月。

[11]   柯蒂斯Carmony，穆章，胡寻超，Abhishek Vasisht Bhaskar和Heng Yin。如果可以，请提取我：滥用PDF

恶意软件探测器中的解析器。在*2016年年报*

*网络与分布式系统安全研讨会（NDSS）*，加州圣地亚哥，2016年2月。

[12]   黎明陈和Algirdas Avizienis。N版本编程：软件操作可靠性的容错方法。在*容错计算，1995*年6 *月*，1995年。

[13]   Corelan队。利用写作教程第11部分：堆喷洒

揭秘，2011年[。https:](https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified) //www.corelan.be/index.php/ 2011/12/31 [/ exploit-writing-tutorial-part-11-heap spray-demystified 。](https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified)

[14]   Igino Corona，Davide Maiorca，Davide Ariu和Giorgio Giacinto。Lux0R：通过对API参考的判别分析来检测恶意的嵌入JavaScript的JavaScript代码。在2014年*人工智能和安全研讨会（AISec）会议记录中*。

[15]   本杰明·考克斯，大卫·埃文斯，阿德里安Filipi，乔纳森Rowanhill，胡伟，杰克·戴维森，约翰·奈特，映阮TUONG，和

贾森Hiser。N变体系统：一个秘密的框架

安全通过多样性。在*第15届USENIX安全研讨会（安全）的诉讼*，加拿大温哥华，2006年7月。

[16]   斯蒂芬·克兰，安德烈·霍伊斯库，斯特凡·布朗塔勒，佩尔·拉森和迈克尔·弗朗茨。通过动态软件分集阻止Cache Side-Channel攻击 在“ *诉讼程序”中*

*2015年度网络和分布式系统安全研讨会（NDSS）*，加州圣地亚哥，2015年2月。

[17]   daehee87。DEFCON 2014 Polyglot [Writeup，2014。http ：// daehee87.tistory.com/393 。](http://daehee87.tistory.com/393)

[18]   ECMA国际。ECMAScript Language Specification，[2016。http ://www.ecma-international.org/ publications / files / ECMA-ST / Ecma-262.pdf 。](http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf)

[19]   利用数据库。攻击性安全攻击数据库“，2016. [https://www.exploit-db.com 。](https://www.exploit-db.com)

[20]   约瑟夫加德纳和Shishir Nagaraja。恶意软件C＆C检测中机器学习的安全性研究。*ACM Computing Survey（CSUR）*，49（3），2016年9月。

[21]   Dan Goodin。Pwn2Own大屠杀继续作为漏洞利用Adobe Reader，Flash，2013. [https：// arstechnica。com / security / 2013/03 / pwn2own-carnage-continue as-exploits-take-down-adobe-reader-flash 。](https://arstechnica.com/security/2013/03/pwn2own-carnage-continues-as-exploits-take-down-adobe-reader-flash)

[22]   Brian Gorenc，AbdulAziz Hariri和Jasiel Spelman。滥用Adobe Reader的JavaScript API。在2015年8月在内华达州拉斯维加斯举行*的第23届DEF CON会议论文集中*。

[23]   Marco Grassi。[CVE-2016-4673]苹果CoreGraphics macOS / iOS JPEG内存损坏，2016。

[https://marcograss.github.io/security/apple/ cve / macos / ios / 2016/11/21 / cve-2016-4673-apple coregraphics.html 。](https://marcograss.github.io/security/apple/cve/macos/ios/2016/11/21/cve-2016-4673-apple-coregraphics.html)

[24]   Petr Hosek和Cristian Cadar。通过Multiversion执行安全的软件更新。在*第35届国际软件工程会议（ICSE）*，加利福尼亚州旧金山，2013年5月。

[25]   卡巴斯基。     卡巴斯基    安全      公告，      2015年[https://securelist.com/files/2014/12/Kaspersky 安全-公告- 2014 EN.pdf 。](https://securelist.com/files/2014/12/Kaspersky-Security-Bulletin-2014-EN.pdf)

[26]   门斯Kolbitsch，本杰明Livshits，本杰明佐恩，和Christian塞弗特。Rozzle：去欺骗互联网恶意软件。在2012年5月在加利福尼亚州旧金山*举行的第33届IEEE安全与隐私专题讨论会（奥克兰）会议记录中*。

[27]   的Pavel Laskov和内迪姆Srndic。恶意JavaScript的PDF文档的静态检测。在*年度计算机安全应用会议（ACSAC）*，2011年。

[28]   Moony Li。黑客团队漏洞发现另一个Windows零日，固定在带外补丁，2015年[。http://blog.trendmicro.com/trendlabs-security intelligence /黑客 - 团队泄漏 - 揭露 - 另一个窗口零日-ms-releases-patch 。](http://blog.trendmicro.com/trendlabs-security-intelligence/hacking-team-leak-uncovers-another-windows-zero-day-ms-releases-patch)

[29]   Daiping Liu, Haining Wang, and Angelos Stavrou. Detecting Malicious Javascript in PDF through Document Instrumentation. In *Proceedings of the 44th International Conference on Dependable Systems and Networks (DSN)*, Atlanta, GA, 2014.

[30]   Kai Lu and Kushal Arvind Shah. Analysis of Use-After-Free Vulnerability (CVE-2016-4119) in Adobe Acrobat and Reader, 2016. [https://blog.fortinet.com/2016/06/06/analysis-ofuse-after-free-vulnerability-cve-2016-4119-inadobe-acrobat-and-reader.](https://blog.fortinet.com/2016/06/06/analysis-of-use-after-free-vulnerability-cve-2016-4119-in-adobe-acrobat-and-reader)

[31]   Xun Lu, Jianwei Zhuge, Ruoyu Wang, Yinzhi Cao, and Yan Chen.

De-obfuscation and Detection of Malicious PDF Files with High Accuracy. In *Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS)*, 2013.

[32]   Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and

Kim Hazelwood. Pin: Building Customized Program Analyis

Tools with Dynamic Instrumentation. In *Proceedings of the 2005*

*ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Chicago, IL, June 2005.

[33]   Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto. A Structural and Content-based Approach for a Precise and Robust Detection of Malicious PDF Files. In *Proceedings of the*

*International Conference on Information Systems Security and Privacy (ICISSP)*, 2015.

[34]   Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto. An Evasion Resilient Approach to the Detection of Malicious

PDF Files. In *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP)*, 2016.

[35]   Davide Maiorca, Igino Corona, and Giorgio Giacinto. Looking at the Bag is not Enough to Find the Bomb: An Evasion of Structural Methods for Malicious PDF Files Detection. In *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Hangzhou, China, March 2013.

[36]   Davide Maiorca, Giorgio Giacinto, and Igino Corona. A Pattern Recognition System for Malicious PDF Files Detection. In *Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, 2012.

[37]   Felipe Andres Manzano. Adobe Reader X BMP/RLE heap corruption, 2012. [http://www.binamuse.com/papers/ XFABMPReport.pdf.](http://www.binamuse.com/papers/XFABMPReport.pdf)

[38]   Net MarketShare. Desktop Operating System Market Share, 2017. [https://www.netmarketshare.com/operatingsystem-market-share.aspx?qprid=10&qpcustomd=0.](https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0)

[39]   Nexor. Preventing Document-based Malware from Devastating Your Business, 2013. [https://www.nexor.com/wpcontent/uploads/2016/02/Preventing-Document-](https://www.nexor.com/wp-content/uploads/2016/02/Preventing-Document-Based-Malware-from-Devastating-Your-Business.pdf)

[Based-Malware-from-Devastating-Your-Business.pdf.](https://www.nexor.com/wp-content/uploads/2016/02/Preventing-Document-Based-Malware-from-Devastating-Your-Business.pdf)

[40]   Nir Nissim, Aviad Cohen, Chanan Glezer, and Yuval Elovici. Detection of Malicious PDF Files and Directions for Enhancements: A State-of-the-art Survey. *Computers & Security*, October 2014.

[41]   Roger Orr. NtTrace - Native API tracing for Windows, 2016. [http://rogerorr.github.io/NtTrace/.](http://rogerorr.github.io/NtTrace/)

[42]   Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson,

          Z. Berkay Celik, and Ananthram Swami.      The Limitations of

Deep Learning in Adversarial Settings. In *Proceedings of the 9th European Workshop on System Security (EUROSEC)*, 2016.

[43]   Babak Salamat, Todd Jackson, Andreas Gal, and Michael Franz. Orchestra: Intrusion Detection Using Parallel Execution and Monitoring of Program Variants in User-space. In *Proceedings of the 4th European Conference on Computer Systems (EuroSys)*, Nuremberg, Germany, March 2009.

[44]   Mark Schloesser, Jurriaan Bremer, and Alessandro Tanasi. Cuckoo Sandbox - Open Source Automated Malware Analysis. In *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, August 2013.

[45]   Florian Schmitt, Jan Gassen, and Elmar Gerhards-Padilla. PDF Scrutinizer: Detecting JavaScript-based Attacks in PDF Documents. In *Proceedings of the 10th Annual International Conference on Privacy, Security and Trust (PST)*, 2012.

[46]   Charles Smutz and Angelos Stavrou. Malicious PDF Detection using Metadata and Structural Features. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2012.

[47]   Charles Smutz and Angelos Stavrou. When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors. In *Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2016.

[48]   Kevin Z. Snow, Srinivas Krishnan, Fabian Monrose, and Niels Provos. ShellOS: Enabling Fast Detection and Forensic Analysis of Code Injection Attacks. In *Proceedings of the 20th USENIX Security Symposium (Security)*, San Francisco, CA, August 2011.

[49]   Kevin Z. Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi.

Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained

Address Space Layout Randomization. In *Proceedings of the*

*34th IEEE Symposium on Security and Privacy (Oakland)*, San Francisco, CA, May 2013.

[50]   Sophps.        The Rise of Document-based Malware,         2016. [https://www.sophos.com/en-us/security-newstrends/security-trends/the-rise-of-documentbased-malware.aspx.](https://www.sophos.com/en-us/security-news-trends/security-trends/the-rise-of-document-based-malware.aspx)

[51]   Alexander Sotirov. Heap Feng Shui in JavaScript. In *Proceedings of the 2007 Black Hat Europe Briefings (Black Hat Europe)*, Amsterdam, Netherlands, 2007.

[52]   Nedim Srndic and Pavel Laskov. Detection of Malicious PDF Files Based on Hierarchical Document Structure. In *Proceedings of the*

*20th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2013.

[53]   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.

[54]   Symantec. Portable Document Format Malware, 2010. [https://www.symantec.com/content/en/us/ enterprise/media/security\_response/whitepapers/the\_rise\_of\_pdf\_malware.pdf.](https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_rise_of_pdf_malware.pdf)

[55]   Symantec.    Internet      Security      Threat         Reports,      2014. [http://www.symantec.com/content/en/us/enterprise/ other\_resources/b-istr\_main\_report\_v18\_2012\_ 21291018.en-us.pdf.](http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf)

[56]   Teryl Taylor, Kevin Z. Snow, Nathan Otterness, and Fabian Monrose. Cache, Trigger, Impersonate: Enabling Context-Sensitive Honeyclient Analysis On-the-Wire. In *Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2016.

[57]   Trend Micro. Macro Malware: Here’s What You Need to Know in 2016, 2016. [http://blog.trendmicro.com/macromalware-heres-what-you-need-to-know-in-2016/.](http://blog.trendmicro.com/macro-malware-heres-what-you-need-to-know-in-2016/)

[58]   Zacharias Tzermias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos. Combining Static and Dynamic Analysis for the Detection of Malicious Documents. In *Proceedings of the 4th European Workshop on System Security (EUROSEC)*, 2011.

[59]   Cristina Vatamanu, Dragos¸ GavriluT¸, and Razvan Benchea. A˘ Practical Approach on Clustering Malicious PDF Documents. *Journal in Computer Virology*, June 2012.

[60]   Kaushik Veeraraghavan, Peter M. Chen, Jason Flinn, and Satish Narayanasamy. Detecting and Surviving Data Races using Complementary Schedules. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, Cascais, Portugal, October 2011.

[61]   Stijn Volckaert, Bart Coppens, and Bjorn De Sutte. Cloning Your Gadgets: Complete ROP Attack Immunity with Multi-Variant Execution. *IEEE Transactions on Dependable and Secure Computing*, 13(4):437–450, July 2016.

[62]   Carsten Willems, Felix C. Freiling, and Thorsten Holz. Using Memory Management to Detect and Extract Illegitimate Code for Malware Analysis. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2012.

[63]   Carsten Willems, Thorsten Holz, and Felix Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. In

*Proceedings of the 28th IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2007.

[64]   Shane Wilton. One Shellcode to Rule Them All: CrossPlatform Exploitation, 2014. [http://www.slideshare.net/ ShaneWilton/one-shellcode-to-rule-them-all.](http://www.slideshare.net/ShaneWilton/one-shellcode-to-rule-them-all)

[65]   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.

[66]   Hui Xue, Nathan Dautenhahn, and Samuel T. King. Using Replicated Execution for a More Secure and Reliable Web Browser. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2012.

[67]   Mark Vincent Yason. Windows 10 Segment Heap Internals. In *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, August 2016.

[68]   Aydan Yumerefendi, Benjamin Mickle, and Landon P. Cox. Tightlip: Keeping applications from spilling the beans. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Cambridge, MA, April 2007.

[[1]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftnref1" \o ") Previous versions of AAR can be obtained from [ftp://ftp.](ftp://ftp.adobe.com/pub/adobe/reader)

[adobe.com/pub/adobe/reader](ftp://ftp.adobe.com/pub/adobe/reader)

[[2]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftnref2" \o ") 2The       samples   are    mainly   obtained      from [http://www.](http://www.pdfscripting.com)

[pdfscripting.com](http://www.pdfscripting.com) and <http://www.planetpdf.com/>

[[3]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftnref3" \o ") VirusTotal labels a sample with CVE number as long as one of the hosted AV products flag the sample with the CVE label.

[[4]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftnref4" \o ") Only CVEs which full details are publicly disclosed are considered

[[5]](file:///E:\\bdfirst\\2018\\2018Q1\\sec17-xu-meng.htm" \l "_ftnref5" \o ") It is worthnoting that PLATPAL cannot be used as the PDF classifier for EvadeML as EvadeML requires a maliciousness score which has to be continuous between 0 and 1 while PLATPAL can only produce discrete scores of either 0 or 1. Therefore, we use PDFrate, the PDF classifier used in the EvadeML paper [65], for this experiment.