在黑暗中变形逃避分类器

                  洪宕                                     尤·哈                                 EE的张简

新加坡国立大学新加坡国立大学新加坡国立大学hungdang@comp.nus.edu.sg a0119416@u.nus.edu changec@comp.nus.edu.sg

抽象

通过对抗数据操作，基于学习的系统已被证明容易逃避。这些攻击已经在假设敌手对目标模型内部结构，训练数据集或至少分配给输入样本的分类评分有一定了解的假设下进行了研究。在本文中，我们调查了一个更加严格和现实的攻击情景，其中目标分类器最低限度地暴露给对手，仅显示其最终分类决策（例如拒绝或接受输入样本）。而且，攻击者只能使用黑箱变形器来操纵恶意样本。也就是说，敌人必须通过在“黑暗中”对恶意样本进行变形来逃避目标分类器。我们提供了一个评分机制，可以根据可用的有限信息为每个样本指定一个真实值评分，该评分反映了每个样本的逃避进度。利用这种评分机制，我们提出了一种逃避方法 - EvadeHC - 并对两款PDF恶意软件探测器（即PDFrate和Hidost）进行评估。实验评估表明，所提议的规避行为是有效的，对评估数据集的规避率为100％。有趣的是，EvadeHC优于已知的基于分类器输出的分类评分运行的分类器规避技术。虽然我们的评估是针对PDF恶意软件分类器进行的，但所提出的方法是与领域无关的，并且更广泛地应用于其他基于学习的系统。

1        引言

机器学习技术在广泛的应用领域中得到了稳定的应用，如图像分类[31]或自然语言处理[33]。已经报道了各种基于学习的系统，以达到最高的准确度，甚至超过人类的表现[15,26]。考虑到训练数据集具有代表性的假设，这些系统预计在运营数据上表现良好。

学习方法也适用于安全任务。在文献[19,22,28]中已经讨论了机器学习在安全环境中的许多创新应用，特别是用于检测安全违规。但是，当基于学习的系统部署用于安全应用时，其准确性可能受到故意噪声和偏差的挑战。各种研究[7,11,12]表明它们可能易于逃避

F:\PDFpy\God_with_me\2018Q2\paper\EvadingClassifiersByMorphingInTheDark\EvadingClassifiersByMorphingInTheDark.files\image001.gif

允许将个人或课堂使用的全部或部分作品的数字或印刷版本免费授予，前提是复制品不是为了获利或商业利益而制作或发行的，并且副本在第一页上包含本通知和全部引用。由ACM以外的其他人拥有的组件的版权必须受到尊重。允许用信用抽象。要复制或重新发布，在服务器上发布或重新分发到列表，需要事先获得特定的许可和/或费用。请求permissions@acm.org的权限。

*CCS '17，2017年10月30日至11月3日，美国德克萨斯州达拉斯* ©2017协会计算机械。ACM ISBN 978-1-4503-4946-8 / 17/10 ... $ 15.00 <https://doi.org/10.1145/3133956.3133978>

通过对抗性数据处理，对于训练数据集的代表性假设，在存在恶意对手时不再成立。例如，有动机的攻击者可能能够变形恶意实例，以使其类似于训练数据集中发现的良性实例，从而逃避目标分类器。

有几篇文章研究了对基于学习的系统的逃避攻击，对攻击者对其所攻击的系统的知识数量做出了不同的假设。例如，Šrndic等人 [30]和谢里夫等人[24] 研究了攻击情景，其中敌手对目标系统的内部知识具有高度的了解（例如，特征空间和分类算法）。Xu等人 [34]调查了一个更有约束的规避方案，其中对手只有对目标检测器进行黑匣子访问，才能为输入样本输出实值分类评分。虽然这些躲避攻击在存在所需辅助信息的情况下有效，有人建议隐藏分类评分的简单预防措施足以阻止这些攻击[34]。因此，这些技术如何与现实中部署的真正黑匣子系统（例如内置于电子邮件服务中的恶意软件检测器）相互作用仍然是一个问题，因为它们不太可能揭示实值分数，而是仅暴露其最终决策（例如，拒绝或接受样本）。

在这项工作中，我们研究了在黑暗中分类器逃避的问题，研究了比现有工作中已经考虑的更严格和现实的回避情景[25,30,34]。特别是，我们研究了一个非常有限的环境，在这个环境中，对手不了解目标系统（例如，模型内部和训练数据集）。而且，目标系统只显示其最终决策，而不是反映其内部状态的实际值分数。我们进一步假设操纵样本的唯一可行方法是通过一些执行“随机”变形的特定工具。

为此，我们制定了一个捕捉上述限制设置的模型。在所提出的模型下，攻击者被限制在三个黑盒，一个敌手试图逃避的二进制输出检测器（或分类器），一个检测样本是否具有恶意功能的测试器以及一个转换样本的变形器。给定恶意样本，攻击者自适应地查询黑盒以搜索避开检测的变形样本。如果样本被目标检测器感知为良性，则样本被认为是逃避检测，但仍保留其恶意功能。

我们证明，在只有黑匣子访问的限制设置下，基于学习的系统仍然容易受到躲避攻击。我们认为这是一个重要的结果，因为它表明建议的防御机制不足以防止对基于学习的系统的逃避攻击。我们通过描述各种逃避方法来证明有效躲避攻击的可行性，其中最主要的是以爬山为基础的接近EvadeHC。EvadeHC的主要组件是一种计分机制，它根据从测试器和检测器获得的二进制输出，为样本分配实际值。直觉就是测量改变探测器和测试者决定所需的变形步骤的数量，并根据这些值推导得分。我们相信这种评分机制可以用来放宽其他设置[25,34]所提供的真实价值评分可用性假设。

我们评估了两种着名的PDF恶意软件分类器（即PDFrate [27]和Hidost [29]）提出的规避技术。评估是在Xu等人以前的工作中使用的数据集上进行的。[34]，以便为基准制定一个公平的基础。该数据集由从Contagio档案库中选择的500个恶意样本组成[6]。我们首先将EvadeHC与基线解决方案进行比较，该解决方案会持续生成随机变形样本，直至找到规避样本。实证结果表明EvadeHC在实验数据集上获得100％的逃避率，并且在执行成本方面胜过基准解决方案高达80倍。此外，我们还对分类器硬化的情况进行了实验（即以增加错误拒绝率为代价降低错误接受率）。结果强有力地证明了EvadeHC的稳健性。我们还将EvadeHC与逃避黑匣子分类器的最新技术进行比较，这些黑匣子分类器在逃避期间使用实值分类得分[34]。有趣的是，即使只能访问检测器的二进制输出，EvadeHC仍然胜过以前的工作。虽然这可能看起来违反直觉，但结果实际上是可以解释的。我们认为原因有两个。首先，EvadeHC基于从测试者和检测者获得的信息做出决定，而不是仅依靠检测器输出的分类得分的先前工作。其次，由我们的评分机制分配的值可以是与真实值分类评分相比更准确的规避逃避困难的指标。

*贡献。*本文做出如下贡献：

（1）  我们给出了一个在黑暗中逃避分类器的公式，因此对手只有黑箱访问探测器，变形者和测试者。我们还给出了一个概率模型HsrMopher来形式化几乎没有领域特定的知识可以在逃避过程中被利用的概念。

（2）  我们设计了一个评分函数，只给出从检测器和测试器获得的二元结果，可以给样本分配反映逃避进展的实值评分。我们认为这种评分机制对于扩展现有的需要分类评分或其他辅助信息的作品是有用的，可以在我们所研究的那种更加有限和实际的环境下进行操作。

（3）  利用评分函数，提出一种有效的爬山式逃避攻击EvadeHC。这种算法是通用的，它不需要关于目标系统的任何知识，甚至不需要关于分类决策的可信度。

（4）  对两种流行的PDF恶意软件分类器进行实验评估。实证结果不仅证明了EvadeHC的效率，而且还证明了其稳健性。更值得注意的是，也有人认为EvadeHC的评分机制比单纯依赖分类评分的评分机制更具信息量[34]。

本文的其余部分的结构如下。接下来，我们通过在第4节中提出概率模型并在第5节中报告实验评估来形式化我们提出的方法。我们讨论了我们规避方法的含义攻击及其缓解策略。我们在第7节结束了第7节的相关工作之前，我们调查了相关工作。

2        问题制定

在本节中，我们制定了在黑暗中逃避分类器的问题并讨论其挑战。在介绍该公式之前，我们举一个例子来说明问题及其相关概念。

2.1        激励情景

让我们考虑一个旨在通过电子邮件向受害者发送恶意软件的对手。对手选择将恶意软件嵌入到PDF文件中，因为受害者比其他文件格式更愿意打开PDF文件。大多数电子邮件服务提供商在传送用户的附件前都会内置恶意软件检测机制。这种恶意软件检测机制通常是一种基于某些提取的特征进行决策的分类器，并使用现有数据对分类模型进行训练。假设攻击者对电子邮件服务提供者使用的检测器没有任何认识（例如，分类器采用的算法和特征空间或其训练数据集）是合理的。尽管如此，攻击者可以通过发送带有恶意负载的电子邮件来探测探测器，可能到它拥有的另一个帐户，并观察来自电子邮件服务器的二进制响应（接受/拒绝）。此外，攻击者可以自适应地修改恶意PDF文件以搜索逃避检测的变体。但是，我们假设对手在被列入黑名单之前，只能对电子邮件服务器的探测器进行有限次数的探测。

对手期望其恶意文件将被检测器拒绝。为了逃避检测，它可以访问两种工具：变换样本的变形工具和测试变形样本是否保留其恶意功能的沙盒工具。例如，变体工具可以在文件的底层表示中插入，删除或替换对象，并且沙盒工具可动态检测样本是否导致易受攻击的PDF阅读器进行某些意外的系统调用。由于对操纵PDF样本的基本机制的理解不足，对手采用变形工具作为黑盒。此外，由于PDF阅读器的复杂性，对手不知道变形的PDF样本是否会保留其功能，

鉴于对检测器和变形机制的知识存在这样的限制，我们的目标是调查是否仍有可能实施有效的攻击。为了捕获这种受限制的能力，我们的公式以三个黑匣子的概念为中心：对手想逃避的二元结果检测器D，“随机”但始终如一地变形样本的Morpher M，以及检查样本功能的检验员T 。

2.2        测试仪T，检测器D和回避

与动机场景中的沙箱相对应的测试者T声明提交的样本*x*是*恶意的*还是*良性的*。T是确定性的，因为它将输出对同一样本的一致决定。

黑盒子检测器D还将样品 *x*作为输入，并决定是*接受*样品还是*拒绝*样品。D对应于激励场景中的恶意软件分类器。被检测器拒绝的样品是被检测器分类为恶意的样品。由T声明的样本*x*有可能被D接受。在这种情况下，我们说样本*x逃避了*检测。当然，如果探测器与测试仪完全相同，那么规避是不可能的。事实上，我们工作的主要目标是研究准确度不够准确的探测器的安全性。与T类似，我们认为检测器是确定性的（即，对于相同的输入，它们的输出总是相同的）。

我们强调，在我们的公式中，检测器的输出是二进制的（例如，接受/拒绝），而不是许多以前的工作（例如，[30,34]）假设实际值输出。这样一个实际值输出可以被解释为其判定中检测器的置信度。

2.3        Morpher M

变形器M将输入样本*x*和随机种子*s*作为输入，并且变形器对应于输出另一样本*x中*描述的变形机制。我们称这种行为是一个变形步骤。该

激励情景。随机种子*小号*供给由变形器所需的随机性，并在同一时间使该变形机制是可复制的（即，给定相同的输入*X*和*小号*中，M一致地输出相同的变形的样品）。我们不关心随机种子*的表示*，为简单起见，将它视为一个简短的二进制字符串。

从样品开始*X* 0 ，对手可以对M个连续的呼叫，以表示的随机种子的序列S =⟨*小号* 1 ，*s ^* 2 ，。。。，*小号大号* ⟩，以获得采样的序列X =⟨*X* 0 ，*X*1 ，*X* 2 ，。。。，*X 大号* ⟩，其中*X 我* = M（*X 我*- 1 ，*小号我*）。让我们称（x，s）为起点*x* 0 ，终点*x L*的*路径*和路径长度*大号*。当从上下文清楚时，我们将在符号中省略s，并简单地用x来引用路径。

该公式没有规定变形如何进行。敌手可以利用应用程序域的有用属性来操纵样本。例如，攻击者可能能够有效地找到连接两个给定样本的变形路径，或知道如何合并两条变形路径以获得某些期望的属性。尽管如此，这些领域特定的属性并不总是可用的。在第4节中，我们提出了一种“随机”变形的概率模型，以捕捉在操纵样本时几乎没有潜在有用特性可以被利用的限制。

2.4        对手的目标和绩效成本

给定一个恶意样本*x* 0 ，攻击者的目标是找到一个以最小代价逃避检测的样本。如果检测器D接受了它，我们称之为样本*回避*，但是展现了由检测器T感知的恶意行为。如果给定样本*x* 0 已经在逃避，那么攻击者已经轻松地达到了目标。否则，攻击者可以调用变形器M来获取其他样本，并通过向T和T发出查询来检查样本

D.

假设*N d*，*N t*和*N m*分别是对手在回避过程中发送给D，T和M的查询次数。我们感兴趣的是*N d*是确定逃税成本的主要因素。在激励情景中，探测器只能被远程访问，而电子邮件服务器（谁是防御者）对这种访问的数量施加了限制。

虽然敌手可以自由访问测试仪，但其计算成本可能不是微不足道的。例如，在激励情景中，需要计算密集的动态分析来检查功能。在我们的实验中，每个这样的测试平均需要大约45秒。另一方面，变形在计算上花费较少。因此，考虑一个目标函数是合理的，其中*N t*比*N m*具有更多的权重。

可能的优化模型是最小化一些成本函数（例如，*N d* + 0.5 *N t* + 0.01 *N m*）。或者，我们可以考虑约束优化，对*N d*施加界限，同时最小化涉及其他两个测量的成本函数（例如*N t* +0.02 *N m*）。由于成本函数取决于应用场景，因此我们不试图将算法调整为任何特定的成本函数。相反，我们设计的搜索策略的总体目标是最小化*N d*，其次是*N t*和*N m*。

2.5        翻转样本和间隙

让我们考虑一个路径X =⟨*X* 0 ，*X* 1 ，。。。，*X 大号*其中⟩ *X* 0 是恶意的，拒绝而*X 大号*是良性和接受。沿途的某个地方，样本从恶意转为良性，从拒绝转为接受。从*x* 0 开始，让我们称第一个样本是*恶意翻转样本*，第一个样本是*拒绝翻转样本*（见图1）。令*m*x和*r*x是从*x* 0开始的变形步骤的相应数量以达到恶意翻转样本和拒绝翻转样本。那是，

*m* x = arg min {T（*x i*）= *良性* }，

*一世*

*r* x = arg min {D（*x i*）= *accept* } *i*

我们呼吁*米*X的*恶意翻转距离*，并且*[R*X的*拒绝翻转距离*。我们通过进一步定义*д*X之间的间隙*米*X和*- [R*X：

*д*X = *- [R*X - *米*X。

值*д*X是特别感兴趣的。如果*д*X <0，则拒绝翻转样品仍然是恶意的，并且因此是一个避让样品。

它通常需要特殊的手工来将恶意代码嵌入恶意样本中。因此，预计一旦样本变得良性，就不太可能通过随机变形恢复恶意功能。对于样本从被接受到被拒绝的类似假设可以做出，对于特征

               恶意的                                  良性

F:\PDFpy\God_with_me\2018Q2\paper\EvadingClassifiersByMorphingInTheDark\EvadingClassifiersByMorphingInTheDark.files\image002.gif

               拒绝                                                          接受

（一个）

                  恶意的                                        良性

F:\PDFpy\God_with_me\2018Q2\paper\EvadingClassifiersByMorphingInTheDark\EvadingClassifiersByMorphingInTheDark.files\image003.gif

                 拒绝                                       接受

（b）中

图1：翻转样品的图示。（a）本maliceflippingsampleis *X* 4 andthereject-flippingsampleis *X* 6 .The恶意翻转距离*米*X = 4和拒绝翻转距离*- [R*X = 6（b）中所述的恶意翻转样品是*X* 5 而thereject-flippingsampleis *X* 3 .Themalice-flippingdistance *米*X = 5以及拒收翻转距离*- [R*X = 3。由于间隙（*[R*X - *米* X ）为负时，*X* 3是在回避。

分类器被训练以识别作为使分类决定不可能通过随机变形形成的理由。因此，我们假设一旦样本从恶意（或拒绝）转变为良性（或被接受），它就不会变形为恶意（或拒绝）。

2.6        在黑暗中逃避的挑战

有限的黑匣子访问存在两种主要的逃避挑战。首先，检测器D只提供接受/拒绝的二进制输出。因此，从一组恶意样本中，不清楚哪一个更接近逃避样本。以前的作品[34]假设D返回一个真实的置信水平，在此基础上规避逃避。尽管如此，在实践中，更谨慎的探测器可能不会透露除最终决定之外的任何信息。

其次，黑箱Morpher M单独不提供操纵样本的灵活性。为了说明，考虑敌手具有恶意样本*x* 0的情况，以及另一个样本*x* 1 丢失其恶意功能并被检测器接受的情况。直觉上，可以从*x* 1 和*x* 0 构造另一个具有较高回避机会的样本*x*。只有黑箱访问M，构建这样的*x*是非常具有挑战性的。一种可能的方法是找到一个样本，该样本位于从*x* 1 开始到*x* 0 结束的路径之间。但是，如何有效地找到连接两个给定样本的变形路径尚不清楚。即使有可能将它们联系起来，沿途样本是否有较高的逃避机会仍然存在一个根本性问题。

3        我们的建议的逃生方法

在本节中，我们提出了逃避方法，该方法可以在检测器拒绝恶意样本的情况下搜索逃避样本。为了说明清楚，我们首先从一个简单的穷举搜索算法SeqRand开始，它作为评估的基准。然后，我们优化SeqRand以降低其运营成本（即向 D和T发布的查询数量），从而获得BiRand。接下来，我们提出一个基于爬山的算法EvadeHC 0，并最终呈现我们主要的逃避方法EvadeHC，它进一步减少了探测器的查询次数。

3.1        逃避通过穷举搜索SeqRand

在被D拒绝的恶意样本*x* 0 上，SeqRand通过连续变形*x* 0 并检查变形的样本与D和T来检查，直到找到规避样本为止。SeqRand是由一个单一的阈值参数化*大号*和进行如下：（1）选择的序列*大号*随机种子S =⟨*小号* 1 ，*s ^* 2 ，。。。，*š 大号* ⟩。

（2）  对于*s*中的每个种子*s i*，通过调用M（*x i*- 1 ，*s i*）获得变形样本*x i*。检查*x i是否*违反D和T，如果它正在逃避，则输出*x i*并停止。否则继续，直到s被消耗。

（3）  如果s已被消耗且没有找到逃避样品，则重复步骤1至3直至找到逃避样品。

令*N r*是当搜索暂停时执行的迭代次数SeqRand。很容易看出发给D，T和M的查询数是相同的：*N d* = *N t* = *N m* = *N r* · *L*

3.2        通过二进制搜索BiRand进行规避

基线方法SeqRand可以通过多种方式进行改进。一个直接的增强是减少每次迭代（aka路径）中发出的探测器查询次数。特别是，在每个变形步骤后不需要调用D，而应该只提交恶意翻转样本之前的样本（例如，图1a中的*x* 3 和图1b中的*x* 4 ）到检测器。此外，还可以通过采用二分查找来代替如基线中的一系列顺序检查来减少用于搜索恶意翻转样本的测试者查询的数量。

算法BiRand在恶意并被D拒绝的输入*x* 0 上执行以下操作：

（1）  选择的序列*大号*的随机种子小号，并与依次调用中号*小号*（从开始*X* 0 ），以获得路径X长度的*大号*。

（2）  查找恶意翻转样品*X*〜使用的binarySearch与测试器T上的路径。设*X*是直接前身*X*〜路径上（ *即* X是恶意的）。

（3）  如果*x*被检测器D接受，则输出*x*并停止; 否则重复步骤1至3，直至找到规避样本。

与SeqRand类似，BiRand也通过单个阈值*L进行*参数化。参数*L*被设置为足够大，使得具有高概率的*L个*变形步骤将使*x* 0 从恶意变为良性。在我们的实验中，*L*范围从50到80。

*查询次数。*我们注意到BiRand在寻找回避样本中所经过的路径数与SeqRand所采用的迭代次数相同，因此它们会产生相同数量的变形步骤。另一方面，BiRand对D和T要求较少的查询。特别地，确定所述恶意翻转各路径上的样品（步骤2）即被至多⌈log 2个*大号*⌉查询T，并且仅存在一个查询以每每个路径（步骤3）的检测器d。总之，我们有：

*Ñ d* = *Ñ - [R*，*Ñ*       *吨* ≤ *Ñ ř* ⌈log 2 *大号* ⌉，*Ñ*        *米* = *ñ - [R* · *大号*

注意，翻转样本的确定被设计为使对D的查询的数量最小化。在*N t*施加比*N d*更大的成本 的情况下，算法可以被修改，以便二进制搜索在D上进行，发现拒绝翻转样品。然后它根据T来测试候选者，每个路径只对T产生一个查询。

*二进制搜索的鲁棒性。*与线性扫描相比，二分查找显着减少了*N t*。正如前面所讨论的那样，我们假设一旦样本变得良性（或被接受），M很可能会将它变形为恶意（或拒绝）。不过，如果发生这种事件，binarysearch可能会返回错误的翻转样本。幸运的是，这种失败的影响仅限于所讨论的道路，并不影响其他途径。

我们可以通过将从D和T获得的信息纳入启发式“引导”逃避的方法来进一步提高逃避效率。为此，我们提出了一种名为EvadeHC 0 的爬山算法 。

3.3        通过爬山避难的 0 回避 0

EvadeHC 0 由两个预先确定的整数*q* 1 ，*q* 2 ，一个预先确定的实际值*跳跃因子* 0 <Δ<1和一个评分函数进行参数化。在迭代算法前进，保持一组*Ç*的 *q* 2名候选人在迭代（在第一次迭代，*Ç*包含给定的恶意样本*X* 0 ）。在每次迭代中，EvadeHC 0 首先生成*q* 1 起始于*C的*起始点的随机路径。如果任何路径包含逃避样本，算法会输出并暂停。否则，从每个路径中选择由跳跃因子Δ确定的新候选者，并且使用以路径翻转距离作为输入的评分函数为每个新候选者计算实数值分数。最后，*q* 2个得分最高的候选者将保留下一次迭代。

每次迭代的细节如下：

（1）  对于每个样品*X*在*Ç*，生成⌊ *q* 1 / |*S*|⌋以*x*为起点的随机路径，获得大约*q* 1个路径。设*P*是生成路径的集合。将*C*重置为空集。

（2）  对于每个路径的x∈*P*，执行以下步骤：

（a）  使用二分搜索查找恶意翻转距离*m*x。

（b）  使用二分查找找出拒绝翻转距离*r*x。如果*m*x > *r*x，则输出拒绝翻转样本并停止（即找到逃避样本）。从起点出发采取步骤。计算得分D

（c）中   设*X*是沿着样品X在⌊Δ的距离*米*X ⌋mor-

*v*=分数（*m*x，*r*x），并将样本*x*与

               分数*v*。插入*X* Dinto *Ç*。                                     d

（3）保持*C中*得分最高的*q*2个样本，并丢弃其余样本。

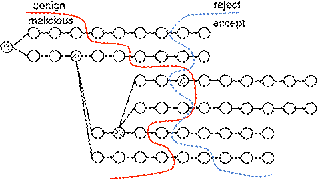


图2：举例说明Hill-Climbing的三次迭代。参数（*q* 1 ，*q* 2 ，Δ）是（2,1,0.5）。因此，*C*只包含一个起始样本。样本*x* 0 ，*x* 1 和*x* 2 分别是为第一次，第二次和第三次迭代选择的起始样本。样本*e*正在逃避。

*评分功能。*一个关键的组成部分是得分功能得分。评分函数的两种可能的选择是基于差或两个翻转距离的相对比率：评分1 （*米*X，*[R*X）= *米*X - *- [R*X，得分 2 （*米*X，*[R*X）= *米*X / *ř*x。

最后，我们介绍我们的主要逃避方法，称为EvadeHC。EvadeHC建立在前面描述的EvadeHC 0 之上。该方法采用分支定界技术来进一步减少所需检测器查询的数量。

3.4        通过登山避难所增强逃避能力

回想一下，在EvadeHC 0中，在每个路径上执行完整的二分搜索（步骤2（b））以找到拒绝翻转距离*r*x。因此，它需要与检测器D进行*q* 1 二进制搜索以在每次迭代中找到最佳的*q* 2个候选者。EvadeHC，另一方面手，进行*单一*的二进制搜索找到一个得分*v*这样恰好*q* 2名考生有分数大于*v*。

具体而言，二分查找采用一组路径*P*作为输入。我们假设*P*中每条路径的恶意翻转距离已经确定。令*v*为目标分数，*P*为达到分数大于*v*的目标路径集合。EvadeHC搜索*v*和*P*在一个单一的二进制搜索，同时保持一组候选路径的*Ç*其最初被设定为*P*。

我们现在描述二进制搜索中的一个步骤，该步骤确定目标分数*v*是否大于测试值*v* 0 。

（1）对于每一个路径X中*Ç*，执行以下内容：

（a）  根据评分函数分数和已知的恶意翻动距离*m*x，找到最小的拒绝翻转距离*r*x'，使得与x相关的分数大于*v* 0 。也就是说，找到最小的*r*x'，使得得分（*m*x - *r*x'）> *v* 0 。

（b）  检查该路径是否达到大于*v* 0 的分数。这是通过查询D来完成的，距离起点距离*r*x'的样本。如果样本被接受，则分数大于*v* 0 。

（2）  如果得分大于*v* 0 的路径总数大于*q* 2 ，则*S*必须大于*v* 0 。舍弃分数低于*v* 0的所有路径，并递归搜索*v* 0 的上半部分。

（3）  否则，*S*必须小于*v* 0 ，所有得分高于*v* 0的路径必须在 *P中*。将它们从*C中*移除，并在*v* 0 的下半部分递归搜索其余路径。

这种减少的适用性取决于评分函数的选择。只要评分函数相对于拒绝翻转距离递减，这是适用的，这对于我们的评分函数得分 1 和得分 2的选择是这种情况。

类似于BiRand，二分查找被设计为减少*N d*。如果对测试者的查询比探测者的查询更昂贵，我们可以在恶意翻转距离上交换和应用二进制搜索，从而减少*N t*。*参数q* 1 ，*q* 2 ，Δ *。*图2说明了搜索过程。直观上，较大的*q* 1 将提高朝向正确方向的准确性，而较大的*q* 2 将通过从局部最小值恢复而增加鲁棒性，并且较大的跳跃因子Δ可以减少好情况下的迭代次数。尽管如此，过大的参数可能不会增加搜索的有效性，但会导致不必要的昂贵的性能成本。我们进一步详细研究了这些参数对EvadeHC性能的影响（第5节）。

4        概率模型

在本节中，我们将拟议的EvadeHC设计背后的直觉形式化。在不失一般性的情况下，为了使搜索有意义，每个样本应该具有基础“状态”的得分值。我们首先讨论一种分配实值状态的机制

（这也是得分）基于T和D的二元结果（“接受”与“拒绝”和“恶意”与“良性”）的样本。接下来，我们提出一个隐藏状态模型来捕捉变形过程似乎是随机的概念。

4.1        国家代表

给定样本*x*，我们从T和D的二元结果中有4种可能的状态。搜索策略通常需要从给定样本集合中选择“最佳”候选者，而这种4状态表示本身不会提供有意义的信息供选择。

我们的主要想法是在理想情况下为从*x*开始的随机路径（由变形器生成）提供一个恶意拒绝翻转样本的概率[[1]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftn1" \o ")。

*作为国家逃避概率。*给定一个起点*x*，让*M x*

是随机路径上恶意翻转距离的随机变量。回想一下，随机路径是随机种子序列产生的。同样，令*R x*和*G x*分别是拒绝翻转距离和间隙的随机变量。*M x*和*R x*不是必须独立的。事实上，人们会期望它们高度正相关，因为检测器试图检测恶意功能。我们将在后面的第5节重新讨论这一点。在这一点上，读者可以参考图11的直觉。该图描绘了源自相同恶意软件的500个变形样本的翻转距离（Pearson相关系数为0 。34）。对于样本*x*，让我们将概率Pr（*G x*<0）指定为其状态。回想一下，具有负面差距的路径意味着它的拒绝翻转样本是恶意的，因此是一个回避样本。换句话说，*x*的状态是随机路径导致逃避样本的概率。现在，假设对手有挑两个候选人中的一个*X*和 *Ÿ*继续搜索，然后用较大的状态值的应试者给出找到一个避让样本的几率较高。这种比较为爬山算法在每一轮中选择候选人提供了一种方法。*预期翻转距离为状态。*明确采用Pr（*G x* <0），因为*x*的状态是在搜索过程中确定概率的高成本。尽管可以通过对多个随机路径进行抽样来估计*G x*的分布，但是这种精确的估计将需要非常多的查询，这反过来抵消了有效搜索策略所提供的收益。或者，我们可以将预期的拒绝翻转和恶意翻转距离作为状态。比较分布*G x*，可以通过绘制更少的随机路径来准确估计期望的距离。在拟议的EvadeHC测得的翻转距离可以看作是预期距离的估计。从预期的距离，我们可以得出预期的差距。直观上，预期差距越小，Pr （*G x* <0 ）的概率就越高。虽然预期的翻转距离不能为我们提供足够的信息来估计概率，但我们仍然可以使用预期的差距来比较概率。这激励了第3.3节中提出的两个得分函数得分 1 和得分 2 的选择。

4.2        隐藏状态随机变形器

我们考虑抽象模型HsrMorpher进行分析。下

HsrMorpher的变形器与散列函数研究中使用的随机Oracle [17]的运行方式相同，为独特查询产生真正随机的结果，并与重复查询保持一致。基于HsrMorpher，我们给出了EvadeHC表现不佳并且不能超越BiRand的条件。

*随机变形的样本。*让我们称之为样品*看出*，如果它已被查询针对T，d或M或已经M的输出。换句话说，如果它已经由黑盒中的一个处理的样品能够被看见。否则样本是 *不可见的*。在由样本*x*和随机种子*s*组成的查询中，M首先检查查询（*x*，*s*）是否已经发出。如果不是这种情况，M随机并一致地选择一个看不见的样本作为变形样本[[2]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftn2" \o ")。否则，将输出先前选择的样本。另外，M还以随机方式决定变形样本的隐藏状态，这将在下面描述。

*随机减少隐藏值。*在HsrMorpher下，每个样本*x*与一个隐藏状态相关联，由两个实数值（*a*，*b*）表示。有一个特定的样本*x o*，其状态（1,1）被设定为最初被看到，并且它也被攻击者知道。在收到前一段。如果*X*是看不见的，则变形器设置stateH查询（*X*，*小号*），M输出仿型样品*X*如在所描述的*X*到（ 0 ， 0 ）并记住。接下来，变形者选择两个实际值（*α*，*β*）从一个随机源R并设置*x* Has 的隐藏状态

（*a* - *α*，*b*- *β*）

其中（*a*，*b*）是*x*的隐藏状态。

当给定查询*x时*，测试者T 根据隐藏状态决定结果。如果*x*不可见，则T将隐藏状态设置为（0,0）。假设的隐藏状态*X*是（*一*，*b*），则T输出*恶意*当且仅当*一个* > 0同样地，对于检测器d，它输出*拒绝*当且仅当*b*>0，并且设置的状态*X*为（ 0 ， 0 ）如果它是不可见的。

请注意，HsrMorpher模型中的唯一参数是随机源R.

4.3        HsrMorpher的EvadeHC分析

HsrMorpher的行为是随机的，并且可以说是对手所处的最坏境况，因为对手无法利用领域知识来操纵样本。例如，给定两个样本*x* 0 和*x* 1 ，不清楚如何找到一个从*x* 0 到*x* 1 变形的随机种子序列。

EvadeHC *无效。*要了解EvadeHC的表现，让我们给一个特定的随机源R HsrMorpher呈现EvadeHC无效。考虑一个随机源R，其中*α*和 *β*是离散的并且只取值0或1.因此，所有可能的隐藏状态是（*a*，*b*），其中*a*，*b*是不大于1的整数。我们可以查看在将EvadeHC作为树执行。不难看出，条件概率Pr（*G x* > 0 |的*状态是*（1,1））对于这棵树中的所有*x*是相同的 。因此，在每次迭代中，所有候选者都具有相同的概率，并且这对下一次迭代而言是无关紧要的。

*随机源* [R *。*HsrMorpher模型假定隐藏值的减少与样本*x*无关。为了验证真实分类器具有这种属性，在我们的实证研究中，我们将分配给样本的内部分类评分视为样本的隐藏状态。实证结果表明，在一些选定的样本中，减少的分布是相似的。

*备注。*人们可以将逃避样本的搜索可视化为减少隐藏状态的比赛（*a*，*b*）。统计上，*a*预计会比*b*下降得更快。逃避的目标是找到一条随机路径，与*b*之前*的*所有可能性相比减少到0 。本质上，EvadeHC通过以下方式实现此目的：它生成并实例化一些随机路径。具有最小差距的路径可以说*b的*速度比平均速度快。因此，EvadeHC沿着这条路径选择一个点作为下一次迭代的起点。图8描绘了基于我们实验中获得的实际痕迹的这个过程。

5        评估

为了研究我们提出的方法的可行性和有效性，我们对两个着名的PDF恶意软件分类器，即PDFrate [27]和Hidost [29]进行了实验评估。我们将PDF恶意软件的概述和两个分类器推迟到附录A.我们首先描述我们的实验设置，然后报告结果。

5.1        实验设置

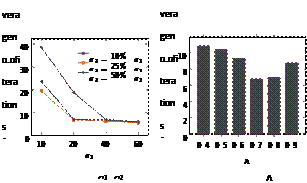
变形器中号。我们采用一种简单的变形机制M，它执行三项基本操作，即插入，删除或替换原始PDF文件中的对象。为了执行这些操作，变形器M需要将PDF文件解析为树状结构，以在这样的结构上执行插入，删除或替换，并最终将修改后的结构重新包装成变体PDF [[3]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftn3" \o ")。我们使用[3]中提供的pdfrw [5]的修改版本进行解析和重新打包。

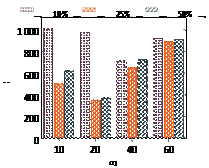
对于插入和替换，放置在原始文件中的外部对象是从良性PDF文件中绘制的。我们通过谷歌搜索收集了大小小于1MB的100个良性PDF文件，作为良性对象的来源输入到M中。它们都被目标检测器接受，并被测试人员T证实没有任何恶意行为。

测试牛逼。测试者T使用称为Cuckoo沙箱的恶意软件分析系统来实现[1]。沙盒在虚拟机中打开提交的PDF文件并捕获其行为，尤其是网络API及其参数。如果PDF文件的行为涉及特定的HTTP URL请求和API跟踪，则该PDF文件被视为恶意文件。我们遵循之前的工作[34]，选择HTTP URL请求和API跟踪作为可靠签名来检测提交的PDF文件的恶意行为。

数据集。我们使用一个由500个恶意PDF文件组成的数据集进行实验研究，这些文件在之前的工作中已经被选定[34]。这些样本来自Contagio [6]数据集，满足以下三个条件。首先，他们表现出测试人员观察到的恶意行为。其次，它们必须与变形机制兼容（即可以通过pdfrw正确重新包装）。最后，他们都被有针对性的探测器拒绝。我们将此称为恶意软件种子。

有针对性的探测器。我们实验中的目标检测器基于两种PDF恶意软件分类器，即PDFrate和Hidost。我们按照之前的作品提供的开源实现重新训练分类器[29,30]。每个分类的训练数据由5 ， 000良性和5 ， 000恶意的PDF文件。这些文件是从Contagio数据集的一个子集中随机选择的，它排除了已经为上述实验数据集挑选的500个恶意软件种子。这是为了解决对手无法访问目标分类器的训练数据的限制。另外，我们对分类器进行小的修改，使得最终输出是二元决策，而不是实值分数。





（c）平均*N d* wrt不同*q* 1 ， *q* 2 图3：*q* 1 ，*q* 2 和Δ对EvadeHC性能的影响

评分功能。在实验研究中采用的得分函数为得分 1 （*m*，*r*）=*m*-*r*，其中*m*是恶意翻转距离，*r*是拒绝翻转距离。

机器和其他参数。所有的实验都是在搭载Intel（R）Xeon（R）CPU E5-2620 v3 @ 2.40GHz处理器和64GB物理内存的Ubuntu 14.04商品机器上进行的[[4]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftn4" \o ")。逃避过程可以并行运行，因为不同执行之间没有依赖关系。在我们的实验中，我们专用一台机器运行D和M，另外九台运行测试仪T。每个这九个机器的同时部署24个的虚拟机，并且能够执行大约2的，每小时000测试器的查询。

我们进行了五组实验。第一组实验检查参数*q* 1 ，*q* 2 和Δ如何影响EvadeHC的性能。第二和第三个实验集分别评估所提出的方法在逃避PDFrate和Hidost检测器方面的有效性。逃避的效率取决于黑匣子查询的数量和总体运行时间。在第四组实验中，我们考虑一个假设的设置，其中检测器被硬化以避免更加困难，并且针对依赖于检测器输出的分类分数的密切相关的工作来指导我们的方法来指导规避[34]。我们通过降低默认邮件分类阈值来模拟检测器的硬化：低于0 。5为PDFrate，Hidost为0以下。最后一组实验验证HsrMorpher模型我们在前面第4节中讨论过。除非另有说明，否则报告的结果是10次获得的平均值。

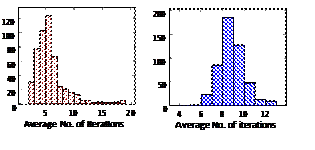


图4：平均迭代次数的直方图EvadeHC需要找到两个避税样本

探测器。

5.2        参数设置对EvadeHC的影响

我们首先研究参数*q* 1 ，*q* 2 和Δ的选择如何影响EvadeHC的性能。我们使用从我们的实验数据集中随机选择的100个恶意软件种子，针对PDFrate检测器运行EvadeHC，使用不同的参数设置，将*q* 1 从10 变化到60，并将*q* 2 与*q* 1 成比例，范围从10％到50％。另外，我们从0 改变跳跃因子Δ 。4至0 。图3中报告了为100种恶意软件种子中的每一种查找逃避样本所需的平均迭代次数和检测器查询量。

图3a表明增加*q* 1 会减少迭代次数。这与我们之前讨论的直觉是一致的，因为较大的*q* 1 会提高算法走向正确方向的可能性。另一方面，*q* 2对迭代次数有更微妙的影响。特别是，太小的值可能导致搜索到局部最小值，阻碍搜索进度，而太大的值可能会意外地将“坏”候选者带到下一次迭代中。在Δ上也观察到类似的趋势（图3b）。当它从0增加时。4至0 。7，我们见证了迭代次数的减少。但是，当它达到0时。如图9所示，该算法倾向于循环更多迭代以找到规避样本。原因在于，在每次迭代中搜索可以进行的Δ限制过小，而不必要的大Δ会增加下一次迭代中样本被拒绝的可能性。

值得注意的是，找到规避样本所需的检测器查询数量不仅取决于迭代次数，还取决于*q* 1 的值。图3c显示了关于*q* 1 和*q* 2的不同设置的检测器查询的平均数量。虽然较大的*q* 1 会导致较少的迭代次数，但它可能会导致每次迭代中检测器查询的数量增加，从而导致整体查询量较大（例如，*q* 2 = 25％*q* 1 ，*q* 1 从20增加到60增加*N d*）。从这个实验集的结果中，我们通过所有其他实验选择*q* 1 = 20，*q* 2 = 5和Δ = 0.75作为EvadeHC的参数设置。

5.3        退出PDFrate检测器

第二个实验集着重于回避PDFrate检测器。我们所有的方法都实现了100％的回避率（即，在我们的数据集中发现了所有500种恶意软件种子的回避样本）。而基线SeqRand和BiRand平均遍历1051个随机路径来查找

（a）*N d*的                                  比率乘以（b）*N t*的比率乘以

SeqRand和BiRand                            SeqRand和BiRand



（c）*N d*与（d）*N t*比BiRand和EvadeHC BiRand和EvadeHC的比率

图5：SeqRand，BiRand和EvadeHC在回避PDFrate方面的性能比较。

一个逃避样本，EvadeHC只需要3到6次迭代（每次迭代涉及评估20个随机路径）。尽管如此，有一些例外要求EvadeHC最多需要19次迭代才能找到逃避样本（图4a）。

我们首先比较BiRand和基线SeqRand 的性能。图5a显示了它们的*N d*的比率，图5b显示了*N t*的比率，清楚地表明了BiRand中采用的二进制搜索机制的有效性。与基线相比，BiRand可以将*N d*降低94倍。

接下来，我们将EvadeHC与BiRand进行比较。图5c报告它们的检测器查询数量之间的比率，而图5d描述了它们的测试器查询的比率。BiRand通常要求2到4倍的探测​​器查询，而EvadeHC的测试查询要多6到10倍。附录B中报告了EvadeHC和BiRand要求的更多黑箱查询详情。

与基准SeqRand相比，EvadeHC的检测器查询次数减少了450倍，测试仪查询次数减少了148次，这意味着运行时间缩短了两个数量级。我们在5.5节报告不同方法的运行时间。

5.4        逃避Hidost检测器

在第三组实验中，我们评估了我们提出的针对Hidost检测器的方法。他们还可以为数据集中的500个恶意软件种子实现100％的回避率。对于95％的种子，EvadeHC在11次迭代中找到了回避样本，而其他种子需要多达13次迭代（图4b）。另一方面，BiRand和SeqRand平均需要经过大约1553条路径才能找到一个逃避样本。

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | F:\PDFpy\God_with_me\2018Q2\paper\EvadingClassifiersByMorphingInTheDark\EvadingClassifiersByMorphingInTheDark.files\image024.gif |  |  |
|  |  | F:\PDFpy\God_with_me\2018Q2\paper\EvadingClassifiersByMorphingInTheDark\EvadingClassifiersByMorphingInTheDark.files\image025.gif |
|  |  |  |

（a）*N d*的                                  比率乘以（b）*N t*的比率乘以

SeqRand和BiRand                            SeqRand和BiRand



（c）*N d*与（d）*N t*比BiRand和EvadeHC BiRand和EvadeHC的比率

图6：SeqRand，BiRand和EvadeHC在逃避Hidost方面的性能比较。

我们报告与SeqRand相比的性能

图6a和图6b中的BiRand，分别显示BiRand在检测器和测试器查询方面能够比SeqRand高出92倍和12倍。

此外，我们还通过显示它们的*N d*（图6c）和*N t*（图6d）之间的比率来对比BiRand和EvadeHC。结果显示EvadeHC优于BiRand。特别是，对于大多数恶意软件的种子，EvadeHC需要两至三倍更少的探测器查询七至十倍少仪查询相比，比兰德。附录C列出了这两种方法所要求的黑箱查询数量的更多细节。

当与基准解决方案进行基准对比时，EvadeHC的执行成本降低了两个数量级，无论是对黑匣子的查询次数还是整个运行时间。

5.5        执行成本

图7报告了逃避两个检测器的不同方法的平均运行时间。如前所述，EvadeHC和BiRand比基线解SeqRand更有效。特别是，为了找到对Hidost的逃避样本，SeqRand平均需要6.7小时，而BiRand和EvadeHC分别只需要40分钟和5分钟。

也可以看出，逃避Hidost通常比逃避PDFrate更昂贵。研究Hidost的构造如何提供抵御攻击的能力将是有趣的。

我们注意到BiRand和EvadeHC的设计前提是*N d*是确定逃逸成本的主要组成部分，因此可最大限度地减少探测器查询的数量，而不是计算成本昂贵的测试器查询。在*N t*是主导组件的情况下（例如，

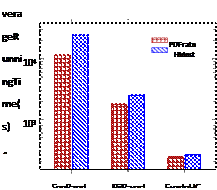


图7：找到不同方法的回避样本所需的平均运行时间（以秒为单位）。结果需要超过5000次规避，其中10次针对我们数据集中的每个恶意软件种子

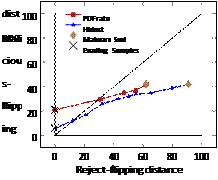


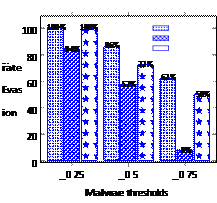
图8：EvadeHC针对两个目标检测器的典型回避曲线。对角虚线代表翻转距离相等的样本。在连续迭代中选择的样本通常会“靠近”线条。

计算代价是主要限制），可以很容易地推导出EvadeHC的一个变体，该变体应用T代替D的单个二分搜索。这种算法的性能可以说与我们当前的实现类似，除了*N d*和*N t*其值将被交换。对准确性没有影响，因为修改纯粹是在算法方面。

5.6        回避痕量分析

为了更好地了解EvadeHC中的爬山启发式如何工作，我们研究了沿典型的回避轨迹产生的样本的翻转距离。规避痕迹是EvadeHC在为恶意软件种子寻找逃避样本时连续生成的样本集合。这样的痕迹从恶意软件种子开始，并以避开样本结束。在第一次迭代中生成的样本可能具有大于恶意翻转距离的拒绝翻转距离，否则已经找到了逃避样本。EvadeHC将从一次迭代中选择有前景的样本，并在下一次迭代中继续对其进行变形，以便它们在丢失其恶意功能之前最终被检测器接受。





（b）强化Hidost

图9：对硬化的探测器采用不同方法（显示在条的顶部）的回避率。

图8描述了典型的回避曲线，这些曲线导致样本不能抵抗PDFrate和Hidost检测器。图中的对角线表示恶意翻转距离和拒绝翻转距离相等的点，垂直线表示检测器的决定从拒绝改变为接受的点，水平线表示样本的恶意功能丢失的点。从直观上看，恶意软件种子在恶意翻转距离和拒绝翻转距离方面的表现常常低于对角线。EvadeHC他们的目标是让它们以横过对角线的方式变形，在横过水平线（即失去恶意功能）之前达到垂直线（即被接受）。从图中还可以看出，与针对Hidost评估的样本相比，针对PDFrate生成的样本通过对角线更容易，这表明PDFrate更加易于使用。

5.7        抗硬化探测器的坚固性

在第四组实验中，我们调查探测器变硬以避免回避更困难的情景。我们通过将恶意软件阈值降低到0来模拟强化的PDFrate 。35 [[5]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftn5" \o ")，并强化了Hidost的-0 。75.此外，我们限制了可以发现的最大检测器查询数量

的避让样品至2 ， 500。如果不能检测器查询的预定数量之后找到的逃避样品，我们处理种子为不可躲避。

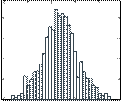
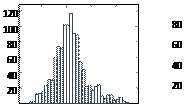
另外，我们用Xu等人的一项技术对两种方法BiRand和EvadeHC进行了基准测试。[34]，我们将其称为EvadeGP。我们强调，尽管EvadeGP依靠分配给样本的实值分类分数来指导搜索，但我们的方法不会假定这些辅助信息。我们根据[34]中报告的一组类似参数运行EvadeGP：群体大小为48，变异率为0 。1和健身停止阈值为0.0。我们将EvadeGP遍历的世代数量限制为50（而不是[34]中的20），从而将每次逃避中发生的探测器查询次数限制为2次，500.我们独立处理每个恶意软件种子，并且不使用适用于一种恶意软件的突变痕迹来搜索其他人的样本。这是为了解决一个非常有限的设置，在这个设置中，攻击者只用一个特制的恶意软件种子开始逃避[[6]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftn6" \o ")。

的图9报告逃避率比兰德和EvadeHC在与该比较EvadeGP靠在硬化探测器。对于恶意软件阈值设置为0的PDFrate检测器。45，EvadeHC和EvadeGP对我们的数据集都达到100％的逃避率，而BiRand只达到91％。当恶意软件阈值进一步降低到0.4和0.35时，EvadeHC的逃避率分别下降到68％和35％，而EvadeGP和BiRand的逃逸率下降幅度更大，达到5％的逃避率比兰德和26％EvadeGP。同样，当Hidost的恶意软件阈值降低到-0.75时， BiRand和EvadeGP分别只能找到8％和50％恶意软件种子的回避样本，而EvadeHC仍然达到62％的回避率（图9b）。

虽然EvadeHC可以比EvadeGP获得更好的逃避率，但即使它只是假设二进制输出而不是EvadeGP的分类得分，我们认为这个结果实际上是可以解释的，但这可能让人感到意外。我们认为原因有两个。首先，EvadeHC的评分机制能够结合从测试者和检测者获得的信息，因此可以说比EvadeGP更具信息性主要依靠检测器输出的分类得分来评分。其次，分类分数并不是样本恶意的可靠表示，事实证明对手可以欺骗分类器来指定“良性分数”来逃避样本。

5.8        验证隐藏状态变形模型

为了证明第4节中讨论的样本状态表示的正确性，我们从同一恶意软件种子生成了200个不同的变形样本，并测量它们的平均翻转距离（拒绝翻转距离是针对Hidost测量的）。每个样本的平均翻转距离是从源自它的20个不同随机路径计算出来的（随机种子的序列对于所有样本是相同的），并绘制在图11中。该图还示出了

        0- 0.2 0           0.2 0.4                     - 0~~0~~ .4 -0.2 0       0.2 0.4

                     （a） *x* 0                                                                           （b） *x* 1

图10：变体对PDFrate分数的影响。直方图报告5个变形步骤后分类分数的变化。

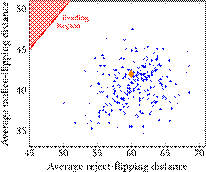


图11：源自相同恶意软件种子（由钻石描绘）的200个变形样本的平均翻转距离（以Hidost作为检测器）。所选恶意软件种子的恶意移动距离为42，拒绝翻转距离为60.皮尔森相关系数为0.34。

恶意并拒绝翻转距离。事实上，距离的皮尔森相关系数为0 。34，在第4节早些时候确认我们的主张，他们是正相关的。

所提出的模型HsrMorpher假设所有样本的隐藏值的减少是独立的和相同的。为了验证现实生活中的分类器具有这种属性，我们考虑PDFrate，并将分配给样本的内部分类分数视为隐藏状态。我们选取一个样本*x* 0，并通过在*x* 0 上应用一系列五个变形步骤来创建另一个样本*x* 1 。然后，我们生成1 ，从长度5的000随机路径*X* 0 。对于每个随机路径，我们记录最后一个样本的分类分数与*x* 0 之间的差异。图10a显示了记录得分差异的直方图。在*x* 1 上进行类似的实验，直方图如图10b所示。两个直方图的相似性证明HsrMoprher是合适的。

我们通过在抽象模型下运行BiRand和EvadeHC来进一步研究HsrMorpher模型的有效性。回想一下，HsrMorpher由随机源*S*参数化，该随机源*S*指示如何选择两个值（*α*，*β*）。从前面的一组实验中，我们观察到典型的恶意翻转距离是42，并且

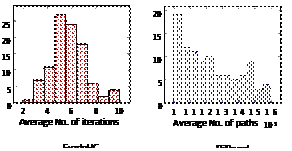


图12：EvadeHC和BiRand在抽象模型下寻找逃避样本所需的迭代次数和路径的平均直方图。

拒绝翻转距离为60（图11）。对经验数据更详细的分析表明，就我们实验中部署的三个黑箱而言，*α*遵循正态分布N（0.024,0.01），*β*遵循N（0.017,0.011）。躲避开始于一个恶意样品*X ö*与状态（ 1 ， 1 ）和成功，如果它可以找到一个逃避样品*ë*与状态（*一*， 0 ）对于一些*一*>0。我们重复实验100次。

由于空间约束，我们只显示了两种方法在HsrMorpher模型下搜索规避样本的平均迭代次数（或路径）（图12）。实验结果证实了HsrMorpher的行为一致性和实证研究。这强烈地表明，抽象模型确实可以作为研究和分析所提议的逃避机制的基础。

6        讨论

在本节中，我们将讨论我们的躲避攻击提出的潜在缓解策略和影响。

6.1        现有的防御机制

逃避分类的大多数现有作品都依靠实值分类来指导逃避[21,34]。因此，有人建议，隐藏分类分数或将随机噪声添加到分数可以保护分类器[8]。另一个提出的防御机制是使用多分类器系统，其分类评分是从不相交的特征集训练的不同模型中随机选取的[10]。我们的研究结果表明，逃避没有任何分类评分的基于学习的分类器仍然是可行的，这使得上述防御机制无效。

6.2        潜在的缓解策略

加强恶意软件阈值。以前的工作[27,34]认为，对于未被对手操纵的原始样本，分类性能（即错误接受/错误拒绝）通常对恶意软件阈值的选择是鲁棒的，分类评分接近任一极端。例如，PDFrate的作者报告称从0调整恶意软件阈值。2至0 。8对准确性的影响可以忽略不计，因为大多数样本的分数都非常接近0或1 [27]。另一方面，我们的实验研究（5.7节）表明，即使恶意软件阈值稍有变化，也会对逃避困难产生重大影响。因此，一种潜在的缓解策略是将阈值设定为更“限制”。这在今后的作品中值得研究。

随机化。另一个潜在的缓解策略是将分类器嵌入一定程度的随机性。特别是，对于分类分数非常接近阈值的样本，分类器可以以某种概率翻转其分类决策。给定一个命题，即大多数样本的分类评分远离阈值，上述翻转机制不会对分类准确性产生重大影响。类似于以前的缓解策略，需要更多的研究来总结这种对策的效果。

识别逃避尝试/样本。在大多数情况下，在向检测器D提出数百次查询后发现回避样本。另外，在回避过程中针对D查询的样本在一定程度上彼此相似。因此，检测逃避企图/样本的一种可能方法是让D记住已被查询的样本，并应用一些基于相似性/基于聚类的技术来识别逃避查询/样本。换句话说，D必须保持有状态。

6.3        防御的回避

具有讽刺意味的是，还可以使用有效的躲避算法来构建更安全的分类器，该分类器对于逃避攻击是强健的。事实上，以前的研究表明，通过在整个培训过程中注入敌对的例子，可以增强基于学习的模型的稳健性[14,21]。特别是，那些保留所需恶意活动的变形样本，尤其是那些被基于学习的系统误分类的样本，可以包含在训练数据集中，用于下一步的训练。我们认为，通过将变形过程制定为随机但可重复的方法，我们的方法能够产生具有多样性的对抗性例子，进一步提高对抗性训练的有效性。

7        相关工作

在许多情况下，从水印[20]到垃圾邮件发现者检测[35]，对探测器的逃避攻击进行了研究。Šrndic等人。[30]基于目标探测器组件的知识，提出了一个逃生场景的分类法，这些知识可用于对手。这些组件包括*培训*

*数据集*和*分类算法*以及它们的参数，以及分类器已经学习的*特征集*。直觉上，探测器的详细知识显着有利于对手进行躲避攻击。

针对基于学习的分类器的各种攻击[9,24,30]假定攻击者对目标系统内部（例如特征集和分类算法）具有较高的知识水平，并且可以直接操纵特征空间中的恶意种子。另一方面，我们的研究调查了一种不采用任何此类知识的限制性对抗模式。

Papernot等人 [21]考虑了对机器学习的黑匣子躲避攻击。他们的技术涉及对目标模型进行逆向工程。特别是，对手必须训练一个替代模型，其行为与目标系统有点相似，然后根据这种替代模型搜索逃避样本。他们表明，这些逃避样本也被目标系统错误分类。显然，这种方法在很大程度上依赖于敌对样本的可转移性。另一方面，我们的解决方案直接搜索针对目标分类器的回避样本，而无需对替代模型进行培训或对样本的可转移性做出任何假设。如果我们将逆向工程方法视为确定分类器的“全局”参数的方法，则相反，

一系列关于对抗性学习的作品[25,32]也假定黑箱可以访问目标系统，但与他们的对抗目标不同。而Tramèr等人。[32]试图提取每个模型参数的确切值，Shokri等人[ [25]有兴趣推断数据记录是否在目标模型的训练数据集中，我们的重点是欺骗目标系统对错误分类的样本进行错误分类。

与我们提出的技术最密切相关的是Xu等人的研究。[34]。区分我们的技术和以前的工作的主要区别在于对目标检测器输出的分类得分的可用性的假设。虽然以前的工作依赖于这些分数来作出决定，但EvadeHC假定来自检测器的二进制输出，并将这些有限的信息与测试器的二进制输出结合起来以指导规避。

8        结论

我们提出了一个限制对手逃避分类器的模型，只有黑匣子访问。这种限制性设置使我们能够研究利用关于领域知识和分类器特定信息（例如模型内部结构和训练数据集）的有限信息的通用回避攻击。一方面，这种模式为后卫提供了对手能力的基准。另一方面，攻击者可以通过将领域知识纳入泛型攻击来获得更有效的躲避攻击。

我们已经描述了EvadeHC，这是一种逃避二进制结果检测器的通用爬山方法。我们的技术假定对检测器和数据处理机制都知之甚少。针对两种PDF恶意软件分类器进行的实验研究已证明了所提出方法的有效性。虽然实验是在恶意软件分类器上执行的，但所提出的技术及其安全性影响可能也会更广泛地应用于其他基于学习的系统。

致谢

我们感谢Amrit Kumar和沉石琦对本文早期版本的有益讨论和反馈。这项研究得到了国家研究基金会，总理办公室，新加坡根据其企业实验室@大学计划，新加坡国立大学和新加坡电信公司的支持。在这项工作中表达的所有意见和研究结果完全是作者的意见和结论必然反映出赞助商的观点。

参考

[1]     克劳迪奥Guarnieri，Alessandro Tanasi，Jurriaan布雷默和马克Schloesser。布谷鸟沙盒：恶意软件分析系统。。[http://www.cuckoosandbox.org/。](http://www.cuckoosandbox.org/)

[2]     CVE详细信息。Adobe Acrobat Reader：漏洞统计。         [HTTP：// WWW。cvedetails.com/product/497/。](http://www.cvedetails.com/product/497/)

[3]     修改pdfrw。[https://github.com/mzweilin/pdfrw。](https://github.com/mzweilin/pdfrw)

[4]     NedimŠrndić和Pavel Laskov。Mimicus：对抗分类器逃避图书馆。 [https://github.com/srndic/mimicus。](https://github.com/srndic/mimicus)

[5]     Patrick Maupin。PDFRW：读取和写入PDF的纯Python库。[https://github.com/pmaupin/pdfrw。](https://github.com/pmaupin/pdfrw)

[6]     Stephan Chenette。恶意文档存档用于签名测试和研究 - Contagio Malware Dump。[http://contagiodump.blogspot.sg/2010/08/。](http://contagiodump.blogspot.sg/2010/08/)

[7]     Marco Barreno，Blaine Nelson，Anthony D Joseph和JD Tygar。2010年

机器学习的安全性。*机器学习*（2010）。

[8]     Marco Barreno，Blaine Nelson，Russell Sears，Anthony D Joseph和J Doug Tygar。机器学习可以安全吗？在*ASIACCS*。

[9]     巴蒂斯塔比吉奥，伊吉诺科罗纳，达维德马奥卡，布莱恩纳尔逊，内迪姆斯坦迪奇，帕维尔拉斯科夫，乔治贾西托和法比奥罗利。逃避攻击

在测试时机器学习。在*ECML-PKDD中*。

[10]   巴蒂斯塔比吉奥，乔治弗梅拉和法比奥罗利。2009用于敌对分类任务的多分类器系统。在*MCS中*。

[11]   巴蒂斯塔·比吉奥，布莱恩·尼尔森和帕维尔·拉斯科夫。2012年。中毒袭击

支持向量机。*arXiv预印本arXiv：1206.6389*（2012）。

[12]   MichaelBrückner，Christian Kanzow和Tobias Scheffer。对抗学习问题的静态预测游戏。*机器学习研究杂志*（2012）。

[13]   Marco Cova，Christopher Kruegel和Giovanni Vigna。检测并分析驱动器下载攻击和恶意JavaScript代码。在*WWW*。

[14]   Ian J Goodfellow，Jonathon Shlens和Christian Szegedy。2014年。解释和

利用对立的例子。*arXiv预印本arXiv：1412.6572*（2014）。

[15]   何开明，张翔宇，任少青，孙健。深入研究整流器：在imagenet分类上超越人类级别的性能。

在*ICCV*。

[16]   Adobe系统公司。PDF参考，第六版，1.23版。

（2006年）。

[17]   乔纳森卡茨和耶胡达林德尔。2014年*介绍到现代密码学*。CRC出版社。

[18]   帕维尔拉斯科夫和内迪姆Šrndić。静态检测恶意JavaScript的PDF文档。在*ACSAC*。

[19]   Kyumin李，詹姆斯Caverlee和史蒂夫Webb。发现社交垃圾邮件制造者：社交蜜罐+机器学习。在*SIGIR*。

[20]   李启明，Ee-Chien Chang。2002.公共水印方案的安全性

二进制序列。在*信息隐藏*。

[21]   Nicolas Papernot，Patrick McDaniel，Ian Goodfellow，Somesh Jha，Z Berkay Celik和Ananthram Swami。2017年。对机器学习的实用黑盒攻击。在*ASIACCS*。

[22]   Konrad Rieck，Philipp Trinius，Carsten Willems和Thorsten Holz。使用机器学习自动分析恶意软件行为。*计算机安全杂志*（2011）。

[23]   Karthik Selvaraj和Nino Fred Gutierres。PDF恶意软件的崛起。

[24]   马哈茂德谢里夫，Sruti Bhagavatula，Lujo Bauer和Michael K Reiter。2016.搭配犯罪：对最先进的脸部识别进行真实和隐秘的攻击。

在*CCS中*。

[25]   Reza Shokri，Marco Stronati和Vitaly Shmatikov。2017.会员推理攻击机器学习模型。在*IEEE标准普尔*。

[26]   大卫银等人。通过深度神经网络和树搜索掌握Go的游戏。*自然*（2016）。

[27]   查尔斯斯莫茨和安杰洛斯塔夫鲁。2012.使用元数据和结构特征进行恶意PDF检测。在*ACSAC*。

[28]   罗宾索默和韦恩帕克森。2010.封闭世界之外：使用机器学习进行网络入侵检测。在*IEEE标准普尔*。

[29]   NedimŠrndić和Pavel Laskov。检测基于分级文档结构的恶意pdf文件。在*NDSS中*。

[30]   NedimŠrndić和Pavel Laskov。2014.实践规避基于学习的分类器：案例研究。在*IEEE标准普尔*。

[31]   Yaniv Taigman，Ming Yang，Marc'Aurelio Ranzato和Lior Wolf。深度报道：缩小面部验证中人类表现的差距。在*CVPR中*。

[32]   弗洛里安Tramèr，范张，Ari Juels，迈克尔K Reiter和托马斯Ristenpart。

通过预测apis窃取机器学习模型。在*USENIX安全中*。

[33]   Oriol Vinyals，ŁukaszKaiser，Terry Koo，Slav Petrov，Ilya Sutskever和Geoffrey Hinton。作为外语的语法。在*NIPS*。

[34]   徐维林，颜延军，戴维埃文斯。2016.自动回避分类器。

在*NDSS中*。

[35]   Chao Yang，Robert Harkreader和Guofei Gu。针对不断演变的Twitter垃圾邮件发送者的实证评估和新设计。*IEEE TIFS*（2013）。

|  |  |
| --- | --- |
| 头 | ％PDF-1.5 |
| 身体 | 1 0 obj << / Count 1 / Kids [7 0 R] / Type / Pages  >> endobj  ···  21 0 obj null endobj |
| CRT | 外部参照  0 22  0000000002 65535 f  ···  ···  0000000394 00000 n |
| 预告片 | 预告片  ···  ···  startxref  1637  %% EOF |

图13：一个PDF文件结构的例子

一个           PDF恶意软件分类器

在本节中，我们将概述可移植文档格式（PDF）和PDF恶意软件，并简要介绍两种检测器，即PDFrate [27]和Hidost [29]。

A.1       PDF

The Portable Document Format (PDF) is a standardised file format that is designed to decouple the document presentation from the underlying environment platform (e.g., application software, hardware or even OSes), thus enabling consistent presentation of the document across different platforms [16]. A typical PDF file consists of four parts. The first part – *header*– contains the magic number and the version of the format. The second part – *body*– incorporates a set of PDF objects comprising the content of the file. The third part – *cross-reference table*(CRT) – is an index table that gives the byte offset of the objects in the body. The last part – *trailer*– includes references to the CRT and other special objects such as the root subject. We depict an example of a PDF file in Figure 13.

The header, CRT and trailer are introduced with %PDF, xrefand trailer keywords, respectively. Objects in the body of the file may be either *direct*(those that are embedded in another object) or *indirect*. The indirect objects are numbered with a pair of integer identifiers and defined between two keywords objand endobj. Objects have eight basic types which are Booleans (representing *true*or *false*), numbers, strings (containing 8-bit characters), names, arrays (ordered collections of objects), dictionaries (sets of objects indexed by names), streams (containing large amounts of data, which can be compressed) and the null objects. It is worth mentioning that there are some dictionaries associated with special meanings, such as those whose type is JavaScript(containing executable JavaScript code).

A.2       PDF Malwares

Due to its popularity, PDF files have been extensively exploited to deliver malware. In addition, given an increasing number of vulnerabilities in Acrobat readers reported recently [2], the threat of PDF malwares is clearly relevant. The malicious payploads embedded in the PDFs are often contained in JavaScript objects or other objects that could exploit vulnerabilities of a particular PDF reader in use.

A.3      Detectors

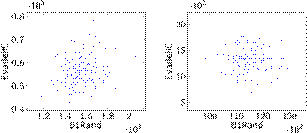
Various PDF malware detectors have been proposed in the literatures. A line of works [13, 18] targeted JavaScript code embedded in the malicious PDFs (or PDF malware). They first extract a JavaScript code from the PDF, and either dynamically or statically analyse such a code to assess the maliciousness of the PDF. Nevertheless, these works would fail to detect PDF malware wherein the malicious payloads are not embedded in JavaScript code or the code itself is hidden [23]. Hence, recent works have taken another approach, relying on the structural features of the PDFs, to perform static detection. Structural feature-based detectors perform detection based on an assumption that there exist differences between internal object structures of benign and malicious PDF files.

Our experimental evaluations are conducted against two stateof-the-art structural feature-based detectors, namely PDFrate [27] and Hidost [29]. These two systems are reported to have very high detection rate on their testing datasets, and are also studied in previous works on evading detection [34]. It is worth mentioning that the outputs of these two detectors are real-value scores which are to be compared against thresholds to derive detection results. In our experiments, we make small changes to their original implementations so that they return binary outputs.

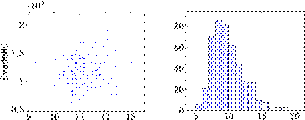
*PDFrate.*PDFrate is an ensemble classifier consisting of a large number of classification trees. Each classification tree is trained using a random subset of the training data and based on an independent subset of features. These features include object keywords, PDF metadata such as author or creation data of the PDF file, and several properties of objects such as lengths or positions. At the time of classification, each tree outputs a binary decision indicating its prediction on the maliciousness of the input. The output of the ensemble classifier is the fraction of trees that consider the input “malicious” (a real-value score raning from 0 to 1). The default cutoff value is set at 0.5.

PDFrate was trained using 5,000 benign and 5,000 malicious PDF files randomly selected from the Contagio dataset [6]. The classifier consists of 1,000 classification trees each of which covers 43 features. The total number of features covered by all the classification trees is 202 (but only 135 are documented in PDFrate’s documentation). An open-source implementation of PDFrate, namely Mimicus [4], is by Šrndić et al. [30]. We utilise this implementation in our experiments.

*Hidost.*Hidost is a support vector machine (SVM) classifier [29]. SVM aims to fit a hyperplane (which can be expressed using a small number of support vectors) to training data in such a manner that data points of both classes are separated with the largest possible margin. Hidost works by first mapping a data point (representing a submitted PDF file) to an indefinite dimensional space using radial



                      (a) *Nd*(b) *Nt*

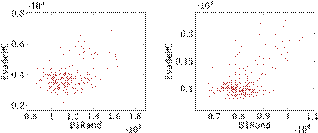


                         BiRand       ·104

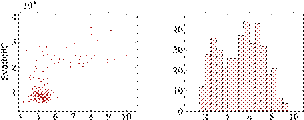
(d) Ratios of *Nm*by

(c)*Nm*BiRand and EvadeHC

Figure 15: Average number of blackbox queries required in evading Hidost



                      (a) *Nd*(b) *Nt*



                         BiRand        ·104

(d) Ratios of *Nm*by

(c)*Nm*BiRand and EvadeHC

Figure 14: Average number of blackbox queries required in evading PDFrate

basis function, and reports the distance between the data point and the hyperplane as a measurement of its maliciousness. If the distance is positive, the PDF file is considered malicious; otherwise, the file is flagged as benign.

Hidost was trained using 5,000 benign and 5,000 malicious PDF files. Hidost operates based on 6,087 classification features, which are structural paths of objects. These structural paths are selected from a set of 658,763 PDF files (including both benign and malicious instances) based on their popularity (i.e., each of them appeared in at least 1,000 files). We use the implementation made available by the author of Hidost in our experiments [29].

B            NUMBER OF BLACKBOX QUERIES IN

EVADING PDFrate

The number of blackbox queries that BiRandand EvadeHCrequired in evading PDFrate are reported in Figures 14a, 14b, 14c. We do not report these metrics of the baseline, for they are all equal to the number of morphing steps that BiRandincurs.

For a majority of the malware seeds, EvadeHCneeds at most 494 queries, while BiRandrequires approximately 1500. The remaining seeds are more difficult to evade, for which EvadeHCand BiRand need up to 786 and 1837 queries, respectively. Similarly, EvadeHC also requires fewer tester queries and less morphing efforts than BiRand(Figures 14b and 14c). In particular, for the majority of the malware seeds, EvadeHCcan find an evading sample with less than 1,237 tester queries and 21,707 morphing steps, while

BiRand consumes up to 9,100 and 63,288 morphing steps.

To get an insight why some seeds necessitated much more effort in finding evading samples, we check their classification scores given by PDFrate. It comes as no surprise that their classification scores are higher than the rest of the malware seeds. In order words, it is harder to find evading samples for these seeds because the detector perceived their maliciousness more clearly.

The histogram of the ratios between the numbers of morphing steps required by the two approaches is depicted in Figure 14d. EvadeHC requires as low as one tenth morphing efforts compared to BiRandin order to find an evading sample.

C            NUMBER OF BLACKBOX QUERIES IN

EVADING HIDOST

We report the amounts of blackbox queries EvadeHCand BiRandincur in evading Hidost in Figure 15a, 15b and 15c. Overall, EvadeHC outperforms BiRandwith respect to all three metrics *Nd*,

*Nt*and *Nm*. In particular, EvadeHC’s requires as few as 427 detector queries, while BiRandneeds as least 1,131 queries to find an evading sample. Regarding the tester, EvadeHCrequires no more than 2073 queries, while BiRandrequires 11,500 queries on average. The similar trend is also observed on morphing effort, wherein EvadeHC requires approximately 10 times fewer morphing steps compared to BiRand(Figure 15d).

[[1]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftnref1" \o ") An alternative choice is to consider the probability that a random path has an evading sample. However, this choice is not suitable, since an arbitrary long path is likely to have an evading sample.

[[2]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftnref2" \o ") If the pool of unseen samples is empty, Msimply halts and does not generate a morphed sample.

[[3]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftnref3" \o ") To avoid redundant parsing of the same PDF file in multiple steps, Mcaches the modified tree structure (or the original tree structure of the malicious PDF file), and directly modifies it without having to parse the PDF file in each step.

[[4]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftnref4" \o ") NCL Testbed. <https://ncl.sg/testbedInformation>

[[5]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftnref5" \o ") Authors of PDFrate [27] reported that adjusting the malware threshold from 0.2 to 0.8 has negligible effect on accuracy.

[[6]](file:///F:\\PDFpy\\God_with_me\\2018Q2\\paper\\EvadingClassifiersByMorphingInTheDark\\EvadingClassifiersByMorphingInTheDark.htm" \l "_ftnref6" \o ") We note that our approaches can trivially support the use of trace replay feature suggested in [34].