使用深度神经网络检测恶意的PowerShell命令

Danny Hendler 本古里安大学 hendlerd@cs.bgu.ac.il
沙伊-凯尔斯-微软 shkels@microsoft.com
阿米尔-鲁宾 本古里安大学 amirrub@cs.bgu.ac.il

摘要

微软的PowerShell是一种命令行外壳和脚本语言，默认安装在Win-dows机器上。基于微软的.NET框架，它包括一个允许程序员访问操作系统服务的接口。虽然管理员可以对Pow- erShell进行配置，以重新限制访问并减少漏洞，但这些限制可以被绕过。此外，PowerShell命令可以很容易地动态生成，从内存中执行，编码和混淆，从而使PowerShell执行的代码的记录和取证分析具有挑战性。

由于所有这些原因，PowerShell越来越多地被网络犯罪分子用作其攻击工具链的一部分，主要用于下载恶意内容和横向移动。事实上，赛门铁克公司最近的一份综合技术报告专门针对网络犯罪分子对Power-Shell的滥用[1]，报告称他们收到的恶意PowerShell样本的数量以及使用PowerShell的渗透工具和框架的数量急剧增加。这说明迫切需要开发有效的方法来检测恶意的PowerShell命令。

在这项工作中，我们通过实现几个新的恶意Power-Shell命令的检测器并评估其性能来应对这一挑战。我们实现了基于 "传统 "自然语言处理（NLP）的检测器和基于字符级卷积神经网络（CNNs）的检测器。使用一个大型的真实世界数据集对检测器的性能进行了评估。

我们的评估结果显示，尽管我们的检测器（尤其是传统的基于NLP的检测器）单独产生了很高的性能，但将基于NLP的分类器和基于CNN的分类器结合起来的合集检测器提供了最好的性能，因为后者的分类器能够检测出成功躲避前者的恶意命令。我们对这些规避命令的分析表明，CNN分类器自动检测到的一些混淆模式在本质上很难用我们应用的NLP技术来检测。

我们的检测器提供了很高的召回值，同时保持了很低的假阳性率，这使我们谨慎乐观地认为它们可以具有实用价值。

1 简介

现代社会比以往任何时候都更加依赖数字技术，医疗保健、能源、交通和银行等重要部门都依赖数字计算机网络来促进其运作。同时，网络犯罪分子和黑客渗透到计算机网络工作中，隐蔽地操纵赌赙数据，或破坏他们的文件并要求支付赎金，风险很大。为了保护不断增长的攻击面，使其免遭坚定而机智的攻击者的攻击，需要开发有效、创新和破坏性的防御技术。

现代网络战争的趋势之一是攻击者对通用软件的依赖。

在被攻击的机器上已经预先存在的工具。微软PowerShell[1]是一种命令行外壳和脚本语言，由于其灵活性、强大的结构和直接从命令行执行脚本的能力，成为许多攻击者的首选工具。一些开源框架，如PowerShell Empire[2]和PowerSploit[3]已经被开发出来，目的是促进PowerShell脚本的开发后网络犯罪的使用。

虽然在检测JavaScript等恶意脚本方面已经做了一些工作[2, 5]3,4,，但Power-Shell尽管在网络战争中地位突出，但学术界相对没有处理。大多数关于PowerShell的工作是由赛门铁克等公司的安全从业人员完成的。
[1]和Palo Alto Networks[6]。这些出版物主要集中在调查PowerShell威胁，而不是开发和评估检测恶意PowerShell活动的方法。缺乏对恶意PowerShell命令的自动检测的研究和基于PowerShell的恶意网络活动的高度流行之间的差异，突出了开发有效的方法来检测这种类型的攻击的迫切需要。

最近在机器学习方面取得的科学成就，特别是深度学习[7]，为开发有效的网络防御的新方法提供了许多机会。由于PowerShell脚本包含文本数据，所以自然要考虑使用自然语言处理（NLP）社区内开发的各种方法来分析它们。事实上，NLP技术被应用于情感分析问题[8]，以及检测恶意的非PowerShell脚本的问题[5]。然而，将NLP技术用于检测恶意脚本并不简单，因为网络攻击者故意混淆其脚本命令以逃避检测[1]。

在NLP情感分析的背景下，深度
将文本视为字符流的学习方法在最近得到了普及，并被证明超越了现有的技术方法。[9, 10]. 据我们所知，我们的工作是第一个提出基于ML（更具体地说，基于深度学习）的恶意Pow-erShell命令的检测器。在最近NLP的字符级深度学习方法的成功激励下，我们也采取了这种方法，考虑到攻击者现有的和未来的混淆企图可能会阻止高级特征的提取，这种方法很有吸引力。

我们开发并评估了几种基于ML的方法来检测恶意的PowerShell程序。其中包括基于新型深度学习架构的检测器，如卷积神经网络（CNN）[11, 12]和循环神经网络（RNN）[13]，以及基于更传统的NLP方法的检测器，如基于字符N-grams和词包的线性分类[14]。

在管理员和开发人员使用的大量良性PowerShell命令中检测恶意的PowerShell命令是一项挑战。 我们使用一个大型数据[4]集来验证和评估我们的检测器，该数据集包括由微软公司网络中的用户执行的60,098条合法的Power-erShell命令和在被各种类型的恶意软件感染的虚拟机上执行的5,819条恶意命令，以及由微软安全专家提供的通过其他方式获得的471条恶意命令。

贡献 我们的工作有两方面的贡献。 首先，我们解决了检测恶意PowerShell命令的重要问题，但研究不足。我们提出并评估了几个基于ML的新型检测器的性能，并在一个大型的真实世界数据集上证明了它们的有效性。

其次，我们展示了字符级的深度学习技术对恶意脚本检测的有效性。我们的评估结果表明，尽管传统的基于NLP的方法产生了较高的检测性能，但将传统的NLP模型与深度学习模型相结合的集合学习，通过检测成功的恶意指令，进一步提高了性能。

[1]https://docs.microsoft.com/en-us/powershell/

[2]https://www.powershellempire.com/

[3][4]https://github.com/PowerShellMafia/PowerSploitUser 敏感数据是匿名的。

逃避传统的NLP技术。

由于字符级的深度学习方法在本质上是独立于语言的，我们希望它可以很容易地适用于检测其他脚本语言的恶意使用。

本文的其余部分组织如下。在第2节中，我们提供了关于PowerShell的背景，以及它是如何被用作攻击载体的，并介绍了理解我们基于深度学习的检测器所需的一些概念。在第2节中，我们3,描述了我们的数据集，我们如何预处理命令以及我们的训练集是如何构建的。第5节提供了我们的检测器的描述，然后4,是对其性能的评估。第6节调查了主要的相关工作。最后，我们对我们的结果进行了总结，并对未来工作的前景进行了简短的讨论。 8

2 背景介绍

2.1 壳牌石油公司（PowerShell

由微软在2006年推出的PowerShell是一种高度灵活的系统外壳和脚本技术，主要用于任务自动化和配置管理[15]。基于.NET框架，它包括两个部分：一个命令行外壳和一个脚本语言。它提供了对关键的Windows系统功能的全面访问，如Windows Management Instrumentation（WMI）和Com-ponent Object Model（COM）对象。此外，由于它是用.NET编译的，它可以访问.NET程序集和DLLs，允许它调用DLL/程序集功能。这些内置的功能使PowerShell具有许多强大的功能，如从远程位置下载内容，直接从内存执行命令，以及访问本地注册表键和计划任务。关于这些功能的详细技术讨论可以在[16]中找到。

作为典型的脚本语言，PowerShell的指令既可以直接通过命令行执行，也可以作为脚本的一部分。PowerShell的功能被大大扩展了，它使用了成千上万的
的'cmdlets'（命令-小程序），它们基本上是
模块化和可重用的脚本，每个脚本都有自己的指定的功能。许多cmdlet是内置在语言中的（如Get-Process和Invoke-Command cmdlet），但额外的cmdlet可以从外部模块加载以进一步丰富程序员的能力。例如，Get-Process cmdlet，当给出一个可以在Power- Shell执行的上下文中访问的机器名称时，会返回在该机器上运行的进程列表。再比如，Invoke-Command cmdlet会根据参数，在本地或一个或多个远程计算机上执行作为其输入的命令。Invoke-Expression cmdlet提供类似的功能，但也支持评估和运行动态生成的命令。

2.1.1 作为攻击媒介的PowerShell

虽然PowerShell可以由公司的IT部门进行配置和管理，以限制访问并重新杜绝漏洞，但这些限制很容易被绕过，正如赛门铁克关于PowerShell的使用增加的综合报告中所描述的那样[1]。此外，记录PowerShell执行的代码也很困难。虽然记录提供给PowerShell的程序可以通过监控执行这些程序的外壳来完成，但这并不一定能提供检测基于PowerShell的攻击所需的可见性，因为PowerShell程序可能使用外部模块和/或使用动态定义的环境变量来调用程序。

例如，Kovter木马[17]使用简单的、随机生成的看起来很无辜的环境变量来调用一个恶意脚本。在我们的数据集中出现的一个这样的命令是 "IEX $env:iu7Gt"，它调用了一个由 "iu7Gt "环境变量引用的恶意脚本。[5]执行shell的日志只会显示动态解释前的com- mand，但不会提供关于恶意脚本的任何数据。

虽然微软在5.0Windows中通过10引入了PowerShell的日志功能，改善了PowerShell的记录能力。
反恶意软件扫描接口(AMSI)通用于---。

[5]IEX是Invoke-Expression的别名。

由于PowerShell脚本是一个非常复杂的界面[18]，许多绕过它的方法已经公布[19，1]，因此对恶意的PowerShell脚本进行有效的取证分析仍然是一个挑战。

除了取证分析的难度外，恶意软件作者还有其他几个很好的理由来使用PowerShell作为他们攻击的一部分[1]。首先，由于PowerShell默认安装在所有Win-dows机器上，其强大的功能可能会被网络犯罪分子利用，他们通常喜欢使用预先安装的工具来快速开发和保持低调。此外，PowerShell几乎总是被列入白名单，因为它被Win-dows系统管理员善意地使用[16]。

其次，由于PowerShell能够下载远程内容并直接从内存中执行命令，它是进行无文件入侵的完美工具[20]，以躲避现有的反恶意软件工具的检测。最后，正如我们接下来所描述的，有多种简单的方法可以混淆Power- Shell代码。

PowerShell代码混淆 如[1]所述，有许多混淆Power- Shell命令的方法，其中许多是由Daniel Bohannon在2016年实现的，并在他创建的"Invoke-Obfuscation "模块中公开可用[21]。 图1列出了我们在数据中遇到的几个关键混淆方法，并提供了它们的使用实例。现在我们简要地解释一下每一种方法。

1. 由于PowerShell命令不区分大小写，小写和大写字母交替出现在恶意命令中。

2. 命令标志通常可以缩短为其前缀。 例如，将PowerShell命令排除在执行策略之外的"-noprofile "标志可以被缩短为"-nop"。

3. 命令可以使用"-EncodeCommand "开关来执行。 虽然这个功能的目标是提供一种包装对DOS不友好的命令的方法，但它经常被恶意代码用来进行混淆处理。

4. 如前所述，"Invoke-Command "小程序可以评估由字符串表示的Power- Shell表达式，因此可用于执行动态生成的命令。

5. 字符可以用"[char]ASCII-VALUE "表示其ASCII值，然后连接起来以创建一个命令或一个操作数。

6. 命令可以进行base-64编码，然后用"FromBase64String "方法转换为字符串。

7. Base64字符串可以用各种方式（UTF8、ASCII、Unicode）进行编码/解码。

8. 另一种混淆命令的方法是插入Power-Shell无视的字符，如`。

9. 在评估之前，可以使用替换和连接功能对命令字符串进行实时操作。

10. 环境变量的值可以在运行时被连接起来，生成一个字符串，其内容将被执行。

11. 一些恶意软件在每次执行命令时都会生成随机名称的环境变量。

虽然以不同方式编码/表示命令并在运行时动态生成命令的能力提供了更大的编程灵活性，但图1说明，这种灵活性很容易被滥用。正如[1]所观察到的，"这些[混淆]方法可以被组合并递归应用，生成在命令行上被深度混淆的脚本"。

2.2 深度学习

在这一节中，我们提供了深度学习概念和架构的背景，这是理解我们在第二节中提出的基于深度学习的恶意Pow-erShell命令检测器所需要的。 4.

身份证	描述	例子
1	交替使用小写和大写字母	-绕过的执行程序 -隐藏的执行程序 (新的执行程序) SYstEM.NET.wEbCLIeNt）.DOWnLoADFiLE（<removed>） 。
2	使用短旗语	-nop -w hidden -e <removed>
3	使用编码的命令	-编码命令<removed>。
4	使用其字符串调用表达式 代表性	- 调用表达式(("New-Object ('Downloadfile')。 ...
5	使用"[char]"而不是一个字符	... $cs = [char]71; $fn = $env:temp+$cs; ...
6	读取数据库中的数据 64	IEX $s=New-Object IO.MemoryStream([Convert]:.) FromBase64String('<removed>'))。
7	使用UTF8编码	$f=[System.Text.Encoding]::UTF8.GetString（ [System.Convert]:FromBase64String(<removed>')）。 ...
8	插入被PowerShell忽视的字符，如 ：`。	...(new-object -ComObject wscript.shell).Popup(E-mail 。 <removed>@<removed>.com 'n 'nClient。 <removed>"`) ...
9	字符串操作	... $filename.Replace('-','/') $... env:temp + ':' + $name + '.exe ...
10	连接变量的内联方式	$emnuxgy='i'; $jrywuzq='x'; $unogv='e'; ...调用表达式 ($emnuxgy+$unogv+$jrywuzq+" ...);
11	在每个变量中使用一个随机的名字 运行	iex $env:vruuyg

图1：PowerShell混淆方法的例子。

人工神经网络[22, 23]是一系列机器学习模型，其灵感来自于生物神经网络，由一系列相互连接的人工*神经元*组成，*按层*组织。一个典型的人工神经网络由一个*输入层*、一个*输出层*和一个或多个*隐藏层*组成。当网络被用于分类时，输出通常是量化的类别概率。一个*深度神经网络*（DNN）有多个隐藏层。有几个关键的DNN架构，下面的小节提供了关于我们的检测器所使用的那些架构的更多细节。

2.2.1 卷积 神经网络(CNN)

CNN是一种学习架构，传统上被用于计算机视觉[24, 25]。我们将对我们使用的CNN深度网络的主要组成部分进行高层次的描述。

顾名思义，CNN的主要组成部分是一个*卷积层*。为简单起见，假设我们的输入是二维灰度图像，卷积层使用二维$k \times k$"过滤器"（又称"内核"），对于一些整数k。当过滤器在二维输入矩阵上滑动时，其$k \times k$*权重*与输入中相应的$k \times k$窗口之间的点积正在被计算。直观地说，滤波器在二维输入矩阵上滑动时，其k × k权重与输入的相应k × k窗口之间的点积正在被计算。

为了搜索某些特征或模式的出现，我们要对输入的每一个k k窗口x进行计算 x形式上，给定一x个k k滤波器，对于应用该滤波器的输入的每个k k窗口x，我们计算$w^{\top}x + b$，其中w是滤波器的权重矩阵，b是代表计算的线性函数常数项的*偏置向量*。在训练过程中，w的²权重以及b的k值都是被学习的。

滤波器以*步长*在输入上滑动，其大小以像素为单位。对一个单一的滤波器用步长s在轮胎输入上滑动进行上述计算的结果是一个二分位数的输出$(n\ k)/s + 1\ (n\ k)/s + 1$，称为滤波器的 "激活图"。 使用$l$个滤波器，并将它们的激活图堆叠在一起，就得到了卷积层的全部输出，其尺寸为$(n\ k)/s$+(1$n\ k)/s$+ll。

为了在使用多个卷积层时保持网络的非线性特性，在每对卷积层之间添加了一个*非线性层*（又称*激活层*）。非线性层应用一个非线性*激活函数*，如*整流线性单元*（ReLU）函数$f\ (x) = max(0,\ x)$，其特性已被[26]研究，或双曲切线$f\ (x) = tanh(x)$ 函数。

*最大集合*层[27]对神经元进行 "缩小采样"，以进行概括并减少过拟合[28]。它在输入上应用了一个k k*的*窗口，并输出窗口内的最大值，从而将参数的数量减少了k^2的系数。直观地说，卷积层的每个输出神经元代表一个图像特征。这些特征通常通过一个或多个全连接层连接到网络的输出，其中输入和输出之间的权重（在训练过程中学习）决定了每个特征对每个输出类别的指示程度。

辍学层[29]可以用在完全连通的地方。为了从概率上减少过度拟合，在每个训练阶段，输入中的每个节点都以概率p留在网络中，或者以概率1p "退出"（与输出断开）。给定一个概率参数p，在每个训练阶段，输入中的每个节点以概率p留在网络中，或者以概率$1p$ "退出"（并与输出断开）。

也出现在递归神经网络中，接下来介绍。

2.2.2 循环神经网络(RNNs)

RNNs是能够处理数据序列的神经网络，例如文本[30, 31]、语音[32, 3433,]、手写体[35]或视频[36]的递归方式，也就是说，通过重复使用到目前为止的输入来处理新的输入。我们使用一个*由长短时记忆*（LSTM）块组成的RNN网络[37]。每个这样的块由一个存储*隐藏状态*的*单元*组成，能够聚合/汇总在很长一段时间内收到的输入。除了单元之外，LSTM块还包含3个被称为*门*的部件，用于控制和调节进入和离开单元的信息流。粗略地讲，*输入门决定了*新输入被单元使用的程度，*遗忘门*决定了单元保留记忆的程度，而*输出门*控制了单元的值被用来计算块的输出的程度。

在文本分析的背景下，一个常见的做法是在LSTM层之前添加一个*嵌入层*[38, 39]。嵌入层有两个目的。首先，它们降低了输入的维度。其次，它们以一种保留其上下文的方式表示输入。 嵌入层将每个输入标记（通常是一个词或一个字符，取决于手头的问题）转换为一个矢量表示。例如，当采取字符级的方法时，我们可以预期由嵌入层计算的所有数字的表示将是相互接近的向量。当问题受益于单词级别的表示时，*word2vec* [40] em-beddings将每个单词表示为一个向量，这样，在训练模型的文本语料库中具有共同语境的单词将由相互接近的向量表示。

双向RNN（BRNN）网络[41]是一种RNN结构，其中两个RNN层连接到输出，一个按顺序读取输入，另一个按反向顺序读取。直观地说，这使得输出可以根据过去和未来状态的形成来计算。BRNN已经在各个领域找到了成功的应用

[42, 43, 44].例如，在感官分析问题的背景下，当处理一个句子中间的文本时，在句子开头看到的文本以及在句子结尾看到的文本都可能被计算使用。

3 该数据集

我们的工作是基于一个大型的数据集，经过预处理（我们很快就会描述），由不同的66,388PowerShell命令组成，6,290被标记为恶意的和60,098被标记为干净的。恶意数据集的命令属于两种类型。对于训练和交叉验证，我们使用通过在沙盒中执行已知的恶意程序并记录该程序执行的所有PowerShell命令获得的不同5,819命令。对于测试，我们使用了5月份由微软安全专家提供2017,的恶意471PowerShell命令。使用后一种类型的恶意实例来评估我们的检测结果，模拟了一个真实的场景，在这个场景中，检测模型使用在沙盒内产生的数据进行训练，然后应用于普通机器上执行的命令。至于干净的命令，我们从Mi-crosoft收到了5月份在微软公司网络中执行的PowerShell命令集合，这些2017,机器在执行PowerShell命令前30天没有任何恶意软件感染的迹象。清洁的命令被分成48,094训练和交叉验证，以及12,004用于测试。

3.1 预处理和训练集构建

我们实现了一个预处理器，其主要目标是进行PowerShell命令的解码和规范化，以提高检测结果。它还消除了相同（以及 "几乎相同"）的命令，以减少数据泄漏的概率。

首先，为了能够对 "明文 "进行检测，我们的预处理器对使用base-64编码的PowerShell命令进行解码。这种命令由-EncodedCommand标志（或任何以'e'或'E'开头的前缀）识别。所有这些命令都要经过base-64解码，否则它们不会提供有用的检测数据。[6]

接下来，预处理器将命令归一化，以减少数据泄漏的可能性[45]，在我们的环境中，使用几乎相同的命令来训练模型和验证模型可能会导致数据泄漏。事实上，我们在我们的数据集中观察到PowerShell命令只在很小的字符数上有差异。在大多数情况下，这是因为使用了不同的IP地址，或者在其他相同的命令中使用了不同数量/类型的空白字符（例如空格、制表符和换行符）。为了避免这个问题，我们将所有数字替换成星号（'*'），将所有连续的空白字符序列替换成一个空格，然后消除重复的字符。

我们还观察到在我们的数据集中，PowerShell的*大小写等价*命令只在字母的大小写上有所不同（见图1中的条目1）。我们通过确保每个大小写等价类中只有一条命令被用于训练/验证来解决这个问题。我们注意到，之前规定的数据集的尺寸与这个预处理阶段*后*的不同命令的数量有关。

我们的数据集是非常不平衡的，因为干净命令的数量要比恶意命令的数量大一个数量级。为了防止模型对大类的偏见，我们通过将每个用于训练的恶意命令重复8次来构建训练集，这样干净/恶意训练命令的比例为1:1。我们倾向于用这种方式来处理不平衡问题，而不是使用欠抽样，以避免过度拟合的风险，因为当使用少量的例子来训练神经网络时，可能会导致过度拟合。

[6]以base-64或UTF8编码的命令参数（见表1中的第6、7项）不会被解码，因为在这些情况下，封装的命令是可用的，可以被检测器分析。

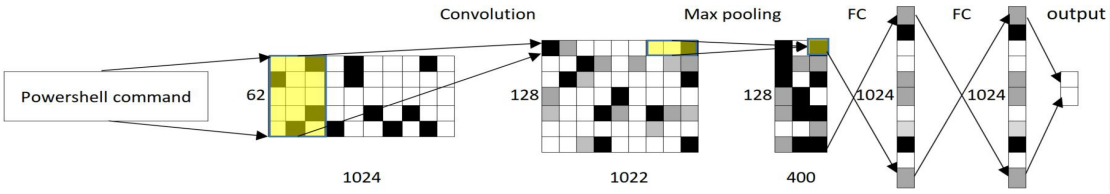


图2：使用的4-CNN架构

4 探测模型

在这一节中，我们描述了我们用于恶意PowerShell命令检测的机器学习模式。然后，我们在本节中评估和比较它们的性能。 5.

我们实现了几个基于深度学习的检测器。为了评估它们在多大程度上能够与更传统的检测方法竞争，我们还实施了基于传统NLP方法的检测器。我们首先描述了这两组模型。

4.1 基于深度学习的检测器

4.1.1 输入准备

神经网络为分类任务进行了优化，其中输入为原始信号[24, 25]。使用这些网络进行文本分类需要对文本进行编码，以便网络能够处理它。Zhang等人[46]探讨了将文本作为 "字符级别的原始信号"，并将其应用于单维CNN进行文本分类。 我们采取了类似的方法，将PowerShell网络指令分类为恶意或良性。

首先，我们选择哪些字符进行编码。我们通过计算每个字符在训练集中出现的次数，然后只给在这些命令中至少出现1.4%的字符分配一个编码。我们将编码阈值设定为这一数值，因为在这一点上，字符的出现频率急剧下降。因此，编码频率最低的字符（也就是
`）出现在大约1.4%的命令中，而最频繁出现的未编码字符（非英语字符）只出现在大约训练集命令的0.3%。

罕见的字符没有被分配一个代码，以减少维度和过度拟合的可能性。结果是一个由61个字符组成的集合，包含空格符号、所有小写英文字母（我们很快会解释如何表示大写字母）和以下符号：-'!%&()*,./:;?@[/] `
{|}+< => û#$^~"

与[46]类似，我们使用的输入特征长度为¯，所以1,024,如果一个命令长于这个长度，它就会被截断。这就减少了网络的尺寸，正如我们在第5.2节中的评估所显示的，这足以提供高质量的分类。然后，CNN网络的输入是通过对命令字符进行 "一键式 "编码来准备的，也就是说，将（可能被截断的）命令的每个字符转换为一个向量，该向量的前61个条目都是0，除了对应于该字符编码的一个条目。所有未被分配代码的字符都被跳过。

在实践中，我们使用62长的向量而不是61长的向量，以处理英语字母的大小写问题。与大多数NLP分类任务不同，在PowerShell命令的上下文中，字母大小写可能是一个强烈的信号（见图1的混淆方法1）。 为了在我们的编码中保留大小写信息，我们增加了一个 "大小写位"，它是第62个向量条目。如果字符是一个大写的英文字母，该位就被设置为1，否则就被设置为0。因此，被输入到CNN网络的PowerShell命令的表示是一个62x1,024的稀疏矩阵。一个代表命令的矩阵如果短于，则用适当数量的零列来1,024填充。

正如我们在第2.2节中所描述的，虽然CNN传统上用于计算机视觉，因此

循环神经网络（RNN）通常接收代表图像的矩阵作为其输入，循环神经网络（RNN）被优化用于处理数据序列。因此，我们提供给RNN分类器的输入是一个大小为1,024的数字向量，其第i个元素是第i个命令字符的代码（如上所述）（未被作为代码的字符被跳过），除了我们明确编码大写的英文字母，因为我们不能使用RNN输入表示的大小写位。

4.1.2 培训

随机梯度下降是训练深度学习模型最广泛使用的方法[47]。我们使用*小批量梯度下降法*来训练我们基于深度学习的算法，其中每个训练*周期*（对训练集的完整传递）被细分为几个*小批量*，这样梯度就会在每个小批量中被计算出来（网络系数也会相应地更新）。

为了在相同的基础上比较我们所有的深度学习网络，在我们所有的实验中，我们使用了16个训练epochs和128个迷你批次的大小。我们还试验了其他数量的epochs/mini-batch，但都没有获得明显的更好的分类结果。

4.1.3 探测模型

我们实施并评估了下文所述的基于深度3学习的检测器。

1. *一个**9**层的**CNN**（**9-CNN**）*。我们使用[46]设计的网络结构，由6个卷积层组成，跨度为1，然后是2个全连接层和输出层。在3个全连接层之间使用了两个滤波层，在第一、第二和最后一个卷积层之后有一个最大集合层。[7]与[46]使用大小为1,024或2,048的全连接层的结构不同，我们在每个这样的层中使用256个条目，因为这在我们的数据上提供了更好的性能。

2. *一个**4**层的**CNN**（**4-CNN**）*。我们还实现了一个较浅的9-CNN架构，其结构如图2所示。它包括一个具有128个大小为62x3的角核的单一卷积层，之后1,是一个3没有重叠的最大集合层。之后是两个全连接层，其大小都是1,024--每个层都有一个概率为0.5的剔除层（图2中未显示），以及一个输出层。

3. ***LSTM***。我们实现了一个由LSTM块组成的递归神经网络工作模型，并使用了上述的字符级表示。由于输入不是自然语言的句子，我们决定不使用Word2Vec
[48] 嵌入。相反，我们的LSTM结构包含一个大小为32的嵌入层。我们使用的LSTM块是双向LSTM单元，输出尺寸为256，然后是两个全连接层，尺寸都是256，使用的辍学概率为 0.5.

4.2 传统的基于NLP的检测器

我们使用了两种NLP特征提取方法--字符级3-gram和词包（BoW）。在这两种方法中，我们评估了tf和tf-idf，然后将逻辑回归分类器应用于所提取的特征上。3-gram模型使用tf-idf的表现更好，而BoW使用 tf表现更好　　。对于每个检测器，我们选择了能提供最佳交叉验证AUC结果的超参数（评估结果见第5节）。请注意，由于4-CNN结构在第一个卷积层中使用了一个长度为3的核，它所使用的特征类似于那些被提取的特征。使用字符级3-gram检测器。

4.3 输入表示的考虑

回顾PowerShell基础的恶意软件作者为避免被发现而使用的混淆方法（见第2.1.1节），我们观察到，我们的输入表示法保留了所需的信息

[7]辍学层和最大集合层通常不计入网络的深度。

来识别它们。用于混淆的命令，包括它们的简短版本（图1中2的混淆方法），由于深度学习模型使用了3个大小的内核，而传统的NLP模型使用了3个语法，因此可以学习到。混淆方法由数据准备期间进行的解码来3解决（见3.1节）。大多数其他混淆方法（见图1）使用特殊字符，如"`"、管道符号""、符号 "+"和环境变量符号"$"。当字符串和环境变量的值在运行时被连接起来进行混淆时，经常使用这些特殊字符。所有这些特殊字符都出现在我们训练集的相当一部分命令中，因此它们都被我们的深度网络输入编码分配了代码。它们也被保留在输入程序中。为传统的NLP模型提供的。

至于随机名称的使用（混淆方法11），这些通常包括数字（与'*'符号相连）或交替的大小写，因此也可以被我们的分类器所学习。(正如我们后面所描述的，我们的深度学习分类器在学习这种模式方面做得更好）。特殊字符串的使用，如"[char]"、"UTF8"、"Base64 "或字符"""，也被两种模型所涵盖，因为它们被保留在输入中。

我们的一些检测器的输入唯一优于其他检测器的混淆方法是交替使用小写/大写字符（图1中混淆方法1）。在我们的CNN深度学习分类器的输入中，大小写位很容易被纳入，而RNN和传统的基于NLP的模型的输入表示则不适合它的使用。

5 评价

我们对训练数据进行了2倍交叉验证，并在表1中列出了我们的检测器的ROC曲线下面积（AUC）结果（四舍五入到小数点后第三位）。除了第4节中介绍的5个检测器外，我们还评估了4-CNN的一个变体（用4-CNN*表示），其中我们没有使用案例位。

-

所有探测器都获得了非常高的AUC水平，范围0是. 9850.990.传统的基于NLP的检测器在.范围98900内提供了很好的结果。990,4-CNN和LSTM检测器略微落后，AUC为0.988，9-CNN提供的AUC较低0。985. 4-CNN*检测器提供的AUC比4-CNN略低，确定案例位是有利的。

对于一个实用的检测器来说，它必须不会产生很多错误的警报。由于网络安全领域的特点是需要分类的事件比率非常高，即使是1%的低假阳性率（FPR）也可能导致太多的假警报。因此，当检测器的阈值被设置为低FPR水平时，评估其提供的真实阳性率（TPR）（又称*召回率*）非常重要。

表中列出2了我们的检测器在FPR水平10^{-2}和训练/交叉验证和测试集上$10^{-3}$$10^{-4}$的TPR。由于我们在测试集中总共有大约12,000个干净的命令，我们在FPR水平为10时停止分析$^{-4}$。表中 "交叉验证 "部分的数值是两个折叠的平均值。测试集 "部分的数值是由在整个训练集上训练的模型得到的。

首先关注交叉验证的结果，可以看出，虽然所有的分类器即使在很低的FPR水平上也能达到很高的TPR值，但传统的NLP检测器的性能更好。3-gram检测器在所有的FPR水平上都处于领先地位，当FPR值降低时，差距会增大。具体来说，即使是1:10,000的FPR，它也能提供0.95的优秀TPR。在基于深度学习的检测器中，4-CNN 和 LSTM 优于 4-CNN* 和 9-CNN 。对于1:10,000的FPR率，4-CNN和LSTM
提供的TPR分别为0.89和0.85。9-CNN在所有实验中获得了最差的结果。

测试集的结果明显较低，但仍然很好。值得注意的是，我们在训练数据上观察到的传统NLP和4-CNN/LSTM模型之间的差距在测试数据上几乎消失了。这似乎表明，后者的模型能够更好地进行概括。

对于1:100的FPR，表现最好的是4-CNN和4-CNN*，TPR为0.89，LSTM次之，为0.88而3-gram和BoW都是最佳。

表1:检测器的ROC曲线下面积（AUC）值。

9-CNN	4-CNN	4-CNN*	LSTM	3克	淘宝网
0.985	0.988	0.987	0.988	0.990	0.989

表2：每个模型的FPR的TPR：交叉验证和测试结果。

FPR	交叉验证测试集			试集		
	10^{-2}	10^{-3}	10^{-4}	10^{-2}	10^{-3}	10^{-4}
9-CNN	0.95	0.89	0.73	0.72	0.52	0.24
4-CNN	0.98	0.96	0.89	0.89	0.76	0.65
4-CNN*	0.97	0.93	0.85	0.89	0.72	0.49
LSTM	0.98	0.95	0.85	0.88	0.81	0.64
3克	0.99	0.98	0.95	0.87	0.83	0.66
淘宝网	0.98	0.93	0.87	0.87	0.50	0.35

87.对于FPR 1:1,000，3-gram检测器是最好的，其TPR为0.83，仅略好于LSTM的0.81 TPR，而对于FPR 1:10,000，所有3-gram、4-CNN和LSTM（按性能递减排序）都能识别大约三分之二的恶意命令0。当比较4-CNN和4-CNN*检测器在FPR水平为1:10,000的测试集上的结果时，案例位的重要性显而易见。使用案例位（4-CNN）时的TPR比不使用案例位（4-CNN*）时的TPR高出近三分之一。9-CNN在测试集的实验中也是表现最差的，比交叉验证测试中的差距更大。

正如我们所提到的，在所有的实验中，测试集的性能都明显低于交叉验证的性能。这是可以预期的：训练集的恶意命令是通过在沙盒内运行恶意软件产生的，而测试集的恶意命令是由安全专家提供的。因此，测试集的恶意命令可能是以不同的方式收集的（例如，通过搜索Windows注册表的恶意Pow-erShell命令），并且可能是由恶意软件产生的，其命令不在训练集中。

5.1 深度/传统模型研究

我们接下来表明，通过结合4-CNN（我们最好的深度学习模型）和3-gram（我们最好的传统NLP模型），我们能够获得比它们各自单独的检测结果更好的检测结果。然后，我们分析了深度模型对传统NLP模型有贡献的恶意命令的类型。

D/T合集的构建方法如下。我们使用4-CNN和3-gram检测器对一个命令进行分类，从而得到两个分数。 如果其中一个分数是0.99或更高，我们就取最高分，否则就取两个分数的平均分。我们以评估非集合算法的方式，通过FPR来评估集合算法在测试集上的TPR性能（见表2）。D/T Ensemble明显优于所有非Ensemble算法，在测试集上获得的TPR分别为0.92、0.89和0.72，FPR水平为1：100、1：1000和1：10000。

为了更好地了解4-CNN检测器对3-gram检测器的贡献，我们在图3a-3c中展示了3-gram、4-CNN和D/T Ensemble检测器对测试集的混淆矩阵。这些结果是用最低的阈值获得的（对每一种算法来说）。

提供不超过10^{-3}的FPR。由于测试集包含大约12,000个干净的实例，这意味着这些算法必须有最多的假12阳性。

通过比较图3a和3c可以看出，D/T Ensemble在3-gram检测器的基础上增加了42个新的检测结果，只有4个新的假阳性。我们分析了这些新的检测结果，以了解深度学习模型能够在哪些方面比传统的NLP模型有所改进。

在新检测到42的命令中，有的15命令包含一串交替出现的数字和字符。在大多数情况下，这个序列代表了主机或域名的名称，该命令是从该主机或域名下载（很可能是恶意的）con- tent。回顾一下，在我们对命令的预处理中，我们把数字转换成星号（见第3.1节），因此主机/域名中含有许多星号。在一个新检测到的命令中出现的这种名称的用法的例子是。".DownloadFile ('http://d*c*a*ci*x*.< domain> ').".

每个人的名字只出现一次，他们
这些名字很可能是由一个域名生成算法（DGA）[49]生成的，该恶意软件用于与其指挥和控制中心进行通信。由于这些名字是唯一的，而且似乎是随机的，3-gram算法无法学习它们的模式，而神经网络则能够学习。

图4a描述了一个例子，说明这样的主机名是如何在神经网络的输入中编码的。请注意在对应于符号 "*"的一行中，零和一交替出现的模式。图4b显示了一个大小为3的神经网络过滤器，它能够检测到这种模式的出现。 该过滤器在对应于 "*"的那一行的第一列和第三列中包含1（预计会在那里发现 "*"符号），在该行的中间一列中包含0，表示这两个数字之间的字符没有意义。当这个滤波器被应用于图4a中描述的字符序列时，它产生了一个相对较强的信号。另一方面，考虑到3-gram的特征提取算法，由于两个数字之间的字符从一个命令变为另一个命令，该模型无法学习这种模式。

类似的论点可以解释D/T Ensemble检测到的一些额外的命令，这些命令没有被3-gram检测到。这些命令包含小写和大写交替的随机字符序列，很可能也是由DGA算法生成的。使用作为其输入的一部分的大小写位，4-CNN能够识别这种模式。

我们注意到，在上述两种情况下，PoweShell命令可能包括额外的恶意迹象，如网络客户端或他们使用的cmdlets。然而，与3-gram检测器不同的是，4-CNN能够检测到包含随机字符和/或大小写的模式，从而使这些com- mand的得分高于阈值。

我们的集合检测器只有7个假阳性（FP），我们对其进行了人工检查。两个FP表现出混淆模式 -- 一个使用[System.Text.Encoding]::UTF8.GetString （在1114条干净的命令中观察到UTF8的使用），另一个使用-EncodedCommand标志（在干净的命令中1,655观察到）。其余五个FP没有使用任何形式的混淆，但它们都至少使用了两个标志，如-NoProfile和-NonInteractive（分别出现在清洁命令5,014中5,833）。

5.2 命令长度的考虑

如前所述，我们的检测器接收的是PowerShell命令的1024个长的前缀，更长的命令被截断了。正如我们的评估所示，这足以在我们的数据集上提供高质量的分类。

未来的恶意软件作者为逃避我们的检测方法而可能采取的反措施是构建长的PowerShell命令，使恶意操作只出现在由无害操作组成的长的无辜前缀之后。在下文中，我们将解释如何挫败这种假想的反措施。

分析我们的数据集的命令长度分布，我们发现86%的恶性命令和88%的干净命令的长度都是一样的

预测结果

实际价值	p	n
pI	373	98
ÃÃÃ	3	12001

(a) 3-gram

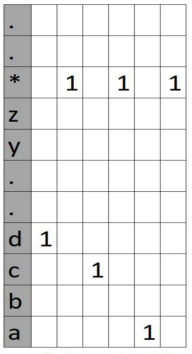
	pn
pI	340	131
ÃÃÃ	5	11999

(b) 4-CNN

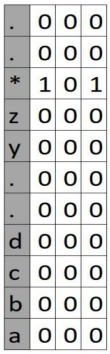
	p	总数
pI	415	56	471
nI	7	11997	12004

(c) D/T组合

图3：3-gram、4-CNN和Ensemble在测试集上的混淆矩阵，使用的阈值导致FPR低于10^{-3}.



名编码样本（为清晰起见去掉了零）。



(b) 能够识别交替出现的数字的过滤器。

图4：一个主机名编码和一个过滤器。网络用于识别交替的数字和函件

是或1,024更少。此外，96.7%的所有恶意软件命令的长度，更重要的是，*99.6%的所有干净的命令的长度是2000或更少*。我们提醒读者，所有的命令都被我们的检测器使用，无论其长度如何--超过1024个字符的命令被简单截断。　鉴于所有检测器的良好性能，我们发现没有理由使用更长的输入尺寸。如果恶意命令的特征发生变化，需要修改我们的检测器以适应2,048更大的输入，这将是非常简单的。到现在为止。

长度超过的清洁命令是2000非常罕见的，认为它们是可疑的。

图中显示5了我们的数据集中良性命令和恶意命令的长度分布，这些命令的长度或1,024更短。恶意命令的长度分布相对向右倾斜，表明恶意的Power-Shell命令往往比良性命令长。非常短的恶意指令的高峰是由于Kovter木马指令[17]，它在我们的数据集中构成了大约8%的恶意指令群。

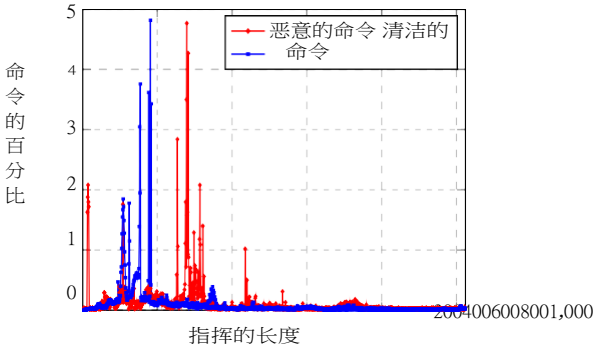


图5：PowerShell命令长度在干净与恶意命令中的分布。

6 相关工作

Zhang等人[46]介绍了一种用于文本分类的深度学习方法，其中对卷积神经网络（CNN）的输入是在字符级别而不是单词级别。他们将其基于深度学习的分类器与基于字的传统NLP方法（如n-grams）和循环神经网络（使用LSTM块）进行了比较。他们的经验评估是使用情感分析和主题分类数据集进行的。他们的结果显示，虽然传统方法在小型/中型数据集上提供了更好的性能，但基于特征的CNN在大型数据集上的性能优于它们。我们的9-CNN架构几乎与他们相同，其输入也是以类似的方式编码的。

Prusa和Khoshgoftaar[50]比较了应用于大型推文数据集的短文情感分析分类的几种架构。他们表明，两个相对较浅的架构（一个包括3个卷积层和2个完全连接的层，另一个包括一个对话层和一个LSTM层）给出了最好的结果。我们的结果与他们一致，在我们的经验评估中，相对较浅的4-CNN网络比较深的9-CNN网络取得了更好的分类性能。 在这两种情况下，分类文本都相对较短--在他们的研究中最多输入140个字符，而在我们的研究中最多输入1,024个字符。

近年来，深度学习方法越来越多地被用于恶意软件检测。其中一些工作（见[51，52，53，54]的几个例子）根据二进制代码和/或其运行时的行为将程序分类为恶意或良性。为了使神经网络能够对可执行程序进行分类，通常需要一个非复杂的特征提取预处理阶段，其结果将被反馈给神经网络。

Athiwaratkun 和 Stokes[54] 使用了一个由Windows可移植可执行文件（PE）组成的大型数据集。他们将深度模型应用于代表这些程序所做的系统调用的输入。他们实施并评估了几个模型，包括一个类似于[46]所使用的字符级CNN。与我们的结果不同，在他们的实证评估中LSTM模型取得了最好的结果。然而，他们的神经网络没有一个是浅层的。

Smith等人也研究了基于PE执行的系统调用的恶意软件检测问题[55]。他们使用了几种分类算法，包括随机森林、CNN和RNN。当输入长度超过1,000个系统调用时，他们发现分类质量有所下降。尽管问题设置和输入领域不同，我们的工作和他们的工作都提供了证据，即通过一些（特定领域的）阈值来限制输入长度可能足以（有时甚至需要）获得良好的性能。

与我们的工作类似，Saxe和Berlin使用深通过分析 "明文 "来检测恶意软件的学习模型[56]。更具体地说，他们将这些模型应用于由（良性和恶意的）URL、文件路径和注册表键组成的大型数据集。他们的CNN架构使用一个单一的卷积层，我们的4-CNN模型也是如此。

尽管之前的一些研究调查了检测恶意脚本语言命令/脚本的问题（可以应用明文分类），但据我们所知，这些研究都没有涉及PowerShell。之前的一些工作通过采用特征提取预处理和浅层分类器的应用，预先发送了恶意JavaScript命令的检测器（见，例如，[2，43,]）。

Wang等人使用深度模型对从网页上收集的JavaScript代码进行分类[5]。与我们的工作类似，他们的模型使用字符级编码，用8位字符表示。他们将他们的分类器与经典的基于特征提取的方法进行了比较，并研究了隐藏层的数量和它们的大小对检测准确性的影响。

近年来，AV厂商发表的一些报告调查并强调了PowerShell作为网络攻击载体的潜在滥用[6, 16, 1]。Pon-tiroli 和 Martinez 分析了恶意PowerShell代码的技术方面[16]。使用真实世界的例子，他们展示了PowerShell和.NET如何被不同类型的恶意软件使用。引自他们的报告。"大量的即用功能使.NET和PowerShell的结合成为可能。

是网络犯罪分子手中的致命工具"。

赛门铁克最近的一份综合技术报告专门讨论了PowerShell被网络犯罪分子滥用的问题[1]，报告称他们收到的恶意PowerShell样本的数量急剧增加，使用PowerShell的渗透工具和框架的数量也在增加。他们还描述了PowerShell命令可以被混淆的许多方法。

总的来说，这些报告揭示了PowerShell在网络攻击的不同阶段可能被使用的方式--从下载恶意内容，到侦察和恶意软件的持续，再到横向移动的尝试。我们在设计检测模型和预处理PowerShell命令时使用了他们提供的一些关于PowerShell攻击的见解。

正如我们之前提到的，微软通过10,引入反恶意扫描接口（AMSI）提高了Windows中PowerShell 5.0的日志记录能力，但许多绕过它的方法已经被公布。这个问题在[19]中得到了讨论和解决，其中PowerShell是建立在.NET架构上的，通过利用.NET的能力来监控PowerShell的活动。正如在他们的工作中所讨论的，所提出的解决方案需要一些调整，这可能会损害PowerShell的性能，以及在机器上产生一些人工制品。

7 讨论

PowerShell命令可以从内存中执行，因此识别恶意命令并在其执行之前阻止它们，一般来说是不现实的。因此，我们估计，我们的检测器最合理的部署场景是作为一个入侵后的工具。在这种部署方案中，执行的PowerShell命令将被记录下来，然后由我们的检测器进行分类。被归类为恶意的命令将产生警报，应该触发进一步的调查。在企业网络中，这种类型的警报通常被发送到安全形成和事件管理（SIEM）系统，并呈现在由组织监控的仪表板上。

该公司的CISO（首席信息安全官）团队。这项工作有几种可以 扩展的方式 。首先，虽然我们已经实现并评估了几个基于深度学习和传统NLP的分类器，但这两类模型的设计空间非常大，对更多的技术和架构进行更全面的评估可以产生甚至更好的检测结果。

其次，在这项工作中，我们的目标是检测通过命令行执行的单个PowerShell命令。未来工作的一个有趣方向是为完整的Power-Shell脚本而不是单个命令设计检测器。这种脚本通常比单个命令长，而且结构更丰富，因为它们通常包含多个命令、函数和定义。对恶意脚本的有效检测可能需要与我们在这项工作中使用的输入编码和/或检测模型明显不同。

未来工作的另一个有趣的途径是，利用微软的反恶意软件扫描接口（AMSI）[18]收集的信息来设计检测器。如前所述，AMSI能够记录在运行时执行的PowerShell命令（静态和动态生成），因此，检测器可能有更多的数据可以操作。然而，尽管AMSI可能不太容易受到第2.1.1节中描述的许多混淆方法的影响，但攻击者可能会找到新的方法来伪装其恶意命令的AMSI痕迹。

8 总结

在这项工作中，我们开发并评估了两类基于ML的恶意PowerShell指令检测器。基于深度学习架构的检测器，如卷积神经网络（CNN）和循环神经网络（RNN），以及基于更传统的NLP方法的检测器，如在字符n-grams和词包之上的线性分类。

我们使用一个大型数据集评估了我们的检测器，该数据集由微软公司网络中的用户执行的合法PowerShell命令、在虚拟机上执行的恶意命令、以及在微软公司网络中的恶意命令。

自由地感染了各种类型的恶意软件，以及由微软安全专家提供的恶意命令。

我们的评估结果表明，我们的检测器产生了很高的性能。最好的性能是由一个集合探测器提供的，该探测器结合了传统的基于NLP的分类器和基于CNN的分类器。我们对能够躲过传统的基于NLP的分类器但被CNN分类器检测到的恶意命令的分析表明，后者自动检测到的一些混淆模式在本质上很难用传统的基于NLP的分类器来检测。我们的集合检测器提供了很高的召回值，同时保持了很低的假阳性率，因此有可能在实践中发挥作用。

参考文献

[1] 赛门铁克。https://www。symantec.com/content/dam/symantec/docs/security-center/whit-papers/ 。increased-use-of-powerhell-in-attacks-16-en. pdf。 2016.

[2] Marco Cova, Christopher Kruegel, and Gio- vanni Vigna.检测和分析驱动式下载攻击和恶意的javascript代码。在*第19届万维网国际会议论文集上*，第281-290页。ACM, 2010.

[3] Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. Zozzle:快速而精确的浏览器内javascript恶意软件检测。在***USENIX**安全研讨会上*，第33-48页。USENIX协会。 2011.

[4] Peter Likarish, Eunjin Jung, and Insoon Jo.使用分类技术检测被掩盖的恶意javascript。In ***Malicious and Unwanted Software (MALWARE), 2009 4th In- ternational Conference on***, pages 47-54.IEEE, 2009.

[5] 王瑶，蔡万东，和魏鹏程.一种用于检测恶意javascript代码的深度学习方法.*安全与通信网络*，9（11）：1520-1534, 2016.

[6] PaloAlto.PullingBack the Cur- tains on EncodedCommand PowerShell Attacks.https://researchcenter.paloaltonetworks.com/2017/03/unit42-pulling-back-the-curtains onencodedcommand-2017.

[7] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville.*深度学习*.自适应计算和机器学习.麻省理工学院出版社。 2016.

[8] Bing Liu and Lei Zhang.观点挖掘和情感分析的调查.In ***Mining text data***, pages 415-463.Springer, 2012.

[9] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 探索语言建模的极限。***arXiv**预印本**arXiv:1602.02410***。 2016.

[10] 张翔和Yann LeCun. 文本未-derstanding from scratch. ***arXiv:1502.01710***。 2015.

[11] Kunihiko Fukushima and Sei Miyake.Neocognitron:一个用于视觉模式识别机制的自组织神经网络模型。In Competition ***and Cooperation in neural nets***, pages 267-285.Springer, 1982.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hin- ton.深度学习.*自然》*，521（7553）：436-444。 2015.

[13] Jeffrey L Elman.在时间中寻找结构。***Cognitive science***, 14(2):179-211, 1990.

[14] Christopher D Manning, Hinrich Schütze, et al. ***Foundations of statistical natural language processing***, volume MIT999. Press, 1999.

[15] 微软公司。 Powershell. https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting？view=powershell-5.1。 2017.

[16] Santiago M Pontiroli和F Roberto Martinez。.net和Powershell恶意软件分析的道。在*Virus Bulletin会议上*。2015.

[17] MicrosoftCorporation. Trojan:win32/kovter. https://www.microsoft.com/en-us/wdsi/threats/ malware-encyclopedia-description?Name=Trojan:win32/Kovter, 2017.

[18] 微软公司。https://msdn.microsoft.com/he-il/library/windows/desktop/dn889587(v=vs.85).aspx.反恶意软件扫描间。2017.

[19] Amanda Rousseau.Hijacking. net to defend powershell. *arXiv preprint arXiv:1709.07508,* 2017.

[20] Michael Hopkins和Ali Dehghantanha.Ex- ploit kits：网络犯罪经济的生产线？在*信息安全和网络取证（InfoSec），2015第二届国际会议上*，第23-27页。IEEE, 2015.

[21] Daniel Bohannon.https://github.com/danielbohannon/Invoke-Obfuscation模式。2016.

[22] Robert J Schalkoff.*人工神经网络*》，麦格劳1.-希尔纽约卷。1997.

[23] B Yegnanarayana.*Artificial neural networks*.PHI Learning Pvt. Ltd.。2009.

[24] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel.应用于手写邮政编码识别的反推法.*神经计算*，1（4）：541-551。1989.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.基于梯度的学习在文档识别中的应用。*IEEE论文集*》，86（11）：2278-2324。1998.

[26] Vinod Nair and Geoffrey E Hinton. 矩形线性单元改善限制性波尔兹曼机器。在第*27届国际机器学习会议（ICML-10）论文集*，第807-814页。Omnipress, 2010.

[27] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. 学习中层特征用于识别。In *Computer Vision and Pat- tern Recognition (CVPR), 2010 IEEE Confer- ence on*, pages 2559-2566.IEEE, 2010.

[28] Douglas M Hawkins.过度拟合的问题.*Journal of chemical information and computer sciences*, 44(1):1-12, 2004.

[29] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov.通过防止特征检测器的共同适应来改进神经网络。*arXiv预印本arXiv:1207.0580*。2012.

[30] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao.用于文本分类的递归卷积神经网络。在*AAAI*，卷页333,2267-2273. AAAI出版社。2015.

[31] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 基于循环租金神经网络的语言模型。在*Interspeech*，卷页2,3.ISCA。2010.

[32] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton.用深度重流神经网络进行语音识别。在*声学、语音和信号处理(icassp)，2013年国际电工委员会国家间会议上*，第6645-6649页。IEEE, 2013.

[33] Alex Graves和Navdeep Jaitly.使用递归神经网络实现端到端的语音识别。在*第31届国际机器学习会议（ICML-14）论文集*，第1764-1772页。JMLR.org, 2014.

[34] Haşim Sak, Andrew Senior, and Françoise Beaufays.用于大规模声学建模的长短时记忆递归神经网络架构。在*第十五届国际语音通信协会年会上*。ISCA, 2014.

[35] Alex Graves.用递归神经网络生成序列。*arXiv预印本arXiv:1308.0850*。2013.

[36] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei.用卷积神经网络进行大规模的视频分类。在*IEEE计算机视觉和模式识别会议上*，第1725-1732页。IEEE, 2014.

[37] Sepp Hochreiter and Jürgen Schmidhuber.长短期记忆。*神经计算*，9（8）：1735-1780。1997.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams.通过反向传播错误学习表征。*自然*，323（6088）：533536,- 1986.

[39] Sam T Roweis and Lawrence K Saul.非线性降维的局部线性EM-Bedding.*科学*，290（5500）：2323-2326, 2000.

[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean.词和短语的分布式表达及其com- positionality.In ***Advances in neural informa- tion processing systems***, pages 3111-3119.NIPS, 2013.

[41] Mike Schuster和Kuldip K Paliwal.Bidirec- tional recurrent neural networks.***IEEE Trans- actions on Signal Processing***，45（11）：2673-2681。1997.

[42] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber.用于改进音素分类和识别的双向LSTM网络。在Wlodzislaw Duch, Janusz Kacprzyk, Erkki Oja, and Slawomir Zadrozny, editors, ***Ar- tificial Neural Networks***。*形式模型及其应用--ICANN第152005,届国际会议，波兰华沙，2005年9月11-15日，会议记录，第二部分，《计算机科学讲座》*第3697卷，第799-804页。Springer, 2005.

[43] Martin Sundermeyer, Tamer Alkhouli, Joern Wuebker, and Hermann Ney.用双向递归神经网络建立翻译模型。在*EMNLP中*，第14-25页。ACL, 2014.

[44] Pierre Baldi, Søren Brunak, Paolo Frasconi, Giovanni Soda, and Gianluca Pollastri.Exploiting the past and the future in protein secondary structure prediction.***Bioinformatics***, 15(11):937 – 946, 1999.

[45] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman.数据挖掘中的泄密问题。Formulation, detection, and avoid- ance.***ACM Transactions on Knowledge Discov- ery from Data (TKDD)***, 6(4): 15, 2012.

[46] Xiang Zhang, Junbo Zhao, and Yann LeCun.用于文本分类的字符级卷积网络.在*神经信息必理系统的进展中*，第649-657页。NIPS, 2015.

[47] Léon Bottou. 大规模机器学习与随机梯度下降。In ***Proceedings of COMPSTAT'2010***, pages 177-186.Springer, 2010.

[48] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.矢量空间中词的有效估计。*arXiv预印本arXiv:1301.3781*。2013.

[49] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. 你的僵尸网络是我的僵尸网络：对僵尸网络接管的分析。在*第16届ACM计算机和通信安全会议上*，第635-647页。ACM, 2009.

[50] Joseph D. Prusa and Taghi M. Khoshgoftaar.用于短文的字符级学习的深度神经网络架构。In ***Proceedings of the Thirtieth International Florida Artificial In- telligence Research Society Conference, FLAIRS 2017, Marco Island, Florida, USA, May 22-24, 2017.***, pages 353-358.AAAI出版社。2017.

[51] Joshua Saxe 和 Konstantin Berlin. 基于深度神经网络的恶意软件检测，使用二维的二进制程序特征。在*恶意和不受欢迎的软件（**MALWARE**），**2015**年第十届国际会议上，第*11-20页。IEEE, 2015.

[52] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu.使用随机投影和神经网络进行大规模的恶意软件分类。In ***Acoustics, Speech and Signal Process- ing (ICASSP), 2013 IEEE International Con- ference on***, pages 3422-3426.IEEE, 2013.

[53] Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas.使用递归网络的恶意软件分类。在*声学、语音和信号处理（**ICASSP**），**2015**年**IEEE**国际会议上*，第1916-1920页。IEEE, 2015.

[54] Ben Athiwaratkun and Jack W Stokes.用lstm和gru语言模型以及字符级cnn进行恶意物品分类.In ***Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on***, pages 248-2486. IEEE, 2017.

[55] Michael R Smith, Joe B Ingram, Christo- pher C Lamb, Timothy J Draelos, Justin E Doak, James B Aimone, and Conrad D James.对可执行文件进行动态分析，以检测和描述恶意软件。 2017.

[56] Joshua Saxe和Konstantin Berlin.揭露：一个带有嵌入的字符级卷积神经网络，用于检测恶意Url、文件路径和注册表键。 2017.