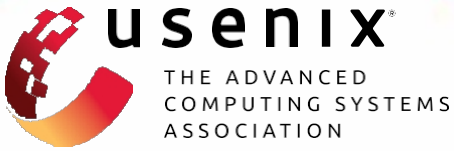

DeepReflect：通过二进制重构发现恶意的功能

Evan Downing，*佐治亚理工学院；***Yisroel Mirsky**，*佐治亚理工学院和本古里安大学；***Kyuhong Park** 和 **Wenke Lee**，*佐治亚理工学院*

https://www.usenix.org/conference/usenixsecurity21/presentation/downing

这篇论文被收录在《第**30**届**USENIX**安全研讨会论文集》中。

开放访问第**30**届**USENIX**安全研讨会的会议记录
是由**USENIX**赞助的。

DeepReflect：通过二进制重构发现恶意的功能

埃文-唐宁
佐治亚理工学院

Kyuhong Park*
佐治亚理工学院

Yisroel Mirsky*
佐治亚理工学院 &
本古里安大学

李文珂
佐治亚理工学院

摘要

深度学习在恶意软件分类方面不断显示出令人鼓舞的结果。然而，为了识别关键的恶意行为，恶意软件分析师的任务仍然是使用静态分析工具对*未知的*恶意软件二进制文件进行逆向工程，这可能需要几个小时。虽然机器学习可以用来帮助识别二进制文件的重要部分，但由于获得足够大的标记数据集的费用，监督方法是不切实际的。

为了提高静态（或手动）逆向工程的生产力，我们提出了DEEPREFLECT：一个在恶意二进制中定位和识别恶意软件组件的工具。为了定位恶意软件组件，我们以一种新颖的方式使用非超视距的深度神经网络，并通过半监督的集群分析对组件进行分类，分析师在日常工作流程中逐步提供标签。该工具是实用的，因为它不需要数据标签来训练定位模型，也不需要最小/非侵入性标签来逐步训练分类器。

在我们与五位恶意软件分析师对超过26000个恶意软件样本的评估中，我们发现DEEPREFLECT将分析师需要逆向工程的功能数量平均减少了85%。我们的方法还能检测出80%的恶意软件组件，而使用基于签名的工具（CAPA）时，只能检测出43%。此外，DEEPREFLECT使用我们提出的自动编码器比SHAP（一种人工智能解释工具）表现更好。这一点很重要，因为SHAP，一个最先进的方法，需要一个标记的数据集，而自动编码器不需要。

1 简介

恶意软件的静态逆向工程可能是一个手动和繁琐的过程。公司每周可以收到多达500万个可移植可执行文件（PE）样本[13]。虽然大多数组织提前对这些样本进行分流，以减少需要分析的恶意软件的数量（即检查VirusTotal [12]的反病毒（AV）引擎结果，在一台电脑上执行样本）。

*这些作者是共同第二作者。

控制的沙箱，提取静态和动态签名，等等），在一天结束时，仍然会有需要静态逆向工程的恶意软件样本。这是由于总是会有新的恶意软件样本，这些样本之前没有被反病毒公司分析过，或者没有被制作成签名来识别这些新样本。最后，样本有可能拒绝在分析员的动态沙箱中执行[42]。

目前的解决方案有创建签名[33,45,72]、分类[14,30,36,41]和对恶意软件样本进行聚类[18,25,52]等形式。然而，这些解决方案只能预测样本的类别（例如，良性与恶意，或一个特定的恶意软件家族）。他们不能定位或解释恶意软件样本本身的行为，而分析师需要执行这些行为来制定报告并改善他们公司的恶意软件检测产品。事实上，由于工作量过大，该领域已经出现了倦怠的报告[27，55]。

为了确定他们的需求，我们咨询了四个反向工程的恶意软件分析师（一个来自AV公司，三个来自政府部门）。我们发现，如果恶意软件分析师有一个工具，可以（1）识别恶意软件中的恶意功能，（2）标记这些功能，那么他们的工作将更有成效。开发这样一个工具的挑战是：（1）需要能够区分什么是良性的，什么是恶意的；（2）理解识别的恶意行为的语义。对于第一个挑战，区分什么是良性的和什么是恶意的是很困难的，因为恶意软件和良性软件的行为往往在很大程度上是重叠的。对于第二个挑战，自动标记和验证这些行为是困难的，因为没有公开的单独标记的恶意软件功能的数据集（不像恶意软件检测和分类系统使用开放的数据集，如防病毒标签）。

为了解决这些挑战，我们开发了DEEPREFLECT，这是一个新颖的工具，它使用(1)一个无监督的深度学习模型，可以定位二进制中的恶意功能和
(2) 一个半监督的聚类模型，该模型使用从以下方面获得的极少的标签对识别的功能进行分类

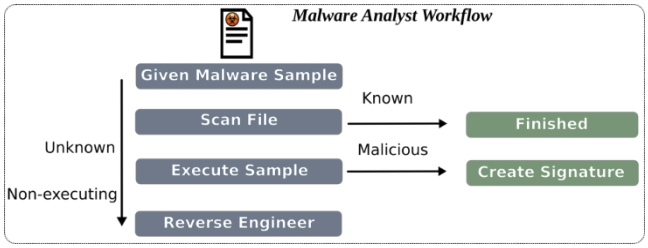

图1：恶意软件分析员的一般工作流程。当分析员必须对一个未知的恶意软件样本进行静态逆向工程时，DEEPREFLECT会协助他们。

分析师的常规日常工作流程。

为了定位二进制中的恶意软件组件，我们使用了一个自动编码器（AE）。AE是一个基于神经网络的机器学习模型，其任务是将其输入重构为其输出。由于网络内层存在压缩，AE被迫学习训练分布中的关键概念。我们的直觉是，如果我们在良性二进制文件上训练AE，它将很难重建恶意的二进制文件（即我们没有训练它的样本）。自然地，AE将无法重建包含恶意行为的二进制文件区域（在良性样本中未见或罕见）。因此，重建错误可以用来识别恶意软件中的恶意组件。此外，由于AE是以无监督的方式训练的，我们不需要数以百万计的标记样本，公司可以利用他们自己的内部恶意软件二进制文件数据集。

为了对定位的恶意软件组件进行分类，我们（1）对恶意软件样本中所有确定的功能进行聚类，（2）使用分析员在其常规日常工作流程中所作的注释对集群进行标注。这种方法是半监督的，因为每个集群只需要几个标签（如三个），就可以给整个集群分配多数标签。随着时间的推移，我们可以通过将AE识别的功能映射到聚类模型中来预测它们的类别（如C&C、权限升级等）。这反过来又节省了分析人员的时间，因为他们不需要被迫一次又一次地对相同的代码进行逆向工程。

我们注意到，无监督的AE为恶意软件分析师提供了直接的效用，而无需训练或使用半监督的聚类模型。这是因为它（1）通过对函数进行排序（按其重建误差），将分析人员的注意力引向最相关的函数；（2）过滤掉那些会花费分析人员数小时或可能数天来解释的函数。

DEEPREFLECT的设计和修订得到了我们四位恶意软件分析师的反馈。然后招募了五个不同的恶意软件分析师来评估DEEPREFLECT的有效性和实用性。总体而言，我们评估了该工具在以下方面的表现：（1）识别恶意软件中的恶意活动、
(2)对相关的恶意软件组件进行聚类，(3)将分析人员的注意力集中在重要的事情上，(4)揭示不同恶意软件家族之间的共同行为，以及

(5) 处理涉及混淆的对抗性攻击。我们的贡献如下：

- 一种新颖的工具，可以帮助恶意软件分析师自动地(1)定位和识别静态恶意软件样本中的恶意行为；(2)通过关联不同恶意软件家族之间的功能关系，得出洞察力。
- 在静态分析中使用机器学习的一种新颖实用的方法，其中
  1. 训练是以无监督的方式进行的：*专家不需要对任何样本进行标记，系统就能产生效用--强调恶意软件的组成部分*；以及
  2. 分类是以半监督的方式完成的，干预程度最低：*分析员常规工作流程中的注释被用作标签，集群中的多数标签被用来对相关的恶意软件组件进行分类*。
- 我们提出了一种方法，通过使用可映射回原始二进制或控制流图的本地化特征，用解释框架（如我们提出的AE或SHAP[40]）对恶意软件的重要部分进行本地化。

2 范围和概述

在这一节中，我们提出了一个激励性的场景，并解释了 我们系统的威胁模型和目标。

2.1 激励

作为一个激励性的例子，让我们假设存在一个名为Molly的恶意软件分析员。她的日常工作流程图可以在图1中找到。这个一般的工作流程是基于最近的工作[69]和我们自己与现实世界的恶意软件分析师的讨论中的描述而现实的。给定一个恶意软件样本，莫莉的任务是了解该样本的作用，以便她能写一份技术报告，并改进她公司目前的检测系统，以便在未来识别该样本。

她首先查询VirusTotal[12]和其他组织，以确定他们以前是否见过这个特定的样本。不幸的是，没有人见过。因此，她进入了下一步，即在一个自定义的沙盒中执行它，以获得样本的动态行为的概况。不幸的是，这个样本并没有显示任何恶意或明显的行为--也有可能是它检测到了环境并拒绝执行。她运行了一些内部工具，试图哄骗恶意软件执行其隐藏行为，但无济于事。在用尽这些方法后，她采取了解包和静态逆向工程的方式来了解其潜在的行为。

在反汇编程序（如IDA Pro[7]或BinaryNinja[1]）中打开解压后的样本时，Molly被其中存在的数千个函数所淹没。她尝试着

运行各种静态签名检测工具来识别恶意软件的一些特定的恶意组件，但同样无济于事。她必须逐一查看每个函数（可能通过其中存在的API调用和字符串进行过滤），试图了解它们的行为（很多时候要借助于调试来验证观察到的行为）。

在注意到它的行为后，她写下她的报告（由妥协指标（IOCs）、静态签名等基本信息组成），并将其传递给她的上级。第二天，她又重复同样的任务。由于这种重复性的手工劳动，这项工作对莫莉来说变得乏味而费时。

DEEPREFLECT旨在通过自动将她的注意力缩小到最有可能是恶意的功能（在她所看到的数千种功能中），并为她在过去看到的那些功能提供标签，从而减轻她费力的工作。

2.2 建议的解决方案

我们提出了DEEPREFLECT，这是一个工具，它（1）在恶意软件二进制中定位恶意功能，（2）描述这些功能的行为。虽然分析人员可能首先试图通过搜索特定的字符串和API调用来静态识别行为[69]，但这些行为很容易被混淆或隐藏起来，无法被分析人员发现。DEEPREFLECT没有这样的假设，它试图通过控制流图（CFG）特征和API调用的组合来识别这些相同的行为。

DEEPREFLECT通过学习良性二进制功能的正常外观来工作。因此，任何异常情况都表明这些功能不会出现在良性二进制文件中，可能被用来促进恶意行为。这使我们的工具能够在分析员打开或扫描二进制文件之前缩小他们的搜索空间。如图5所示，DEEPREFLECT将分析师必须检查的功能数量（在每个恶意软件样本中）平均减少了85%，说明了他们完成任务所需的工作量。此外，我们表明，我们的方法优于基于签名的技术，这些技术，旨在完成相同的目标§4.3。

2.3 威胁模型

我们假设恶意软件分析者正在进行静态分析。静态分析的局限性已在以前的工作中讨论过[44]。我们在本文中不涉及动态分析，尽管从概念上讲，我们的工具可以扩展到动态分析数据的工作。我们假设给我们的系统的恶意软件是无包装的，这与之前的工作[37, 39, 59, 60]类似。之前的工作中已经研究了解包问题，并提出了解决方法[21，58]。我们的结果直接依赖于恶意软件被解包，因此我们依靠先前的工作[11]首先为我们解包二进制文件。我们强调，我们的工具只是分析师管道中的一个步骤，而解包是第一步，如图所示

在图1和图2中。

我们假设我们可以可靠地反汇编恶意软件，以提取基本块和功能。在以前的工作中已经讨论过准确拆解二进制文件的挑战[15，38]。

对于我们的实验，我们相信我们的机器学习模型和数据集是可靠的（也就是说，没有积极地试图攻击或挫败我们的系统）。关于这一假设（及其解决方案）在 部署环境中的局限性的讨论可以在第5.1节中找到。

2.4 研究目标

正如第1节和第2.1节所讨论的，分析人员需要定位和描述恶意软件样本内部功能的行为。因此，DEEPREFLECT有四个主要目标：(*G1*)准确识别恶意软件样本内的恶意活动，(*G2*)在静态分析恶意软件样本时集中分析师的注意力，(*G3*)处理新的（未见过的）恶意软件家族，以及(*G4*)对 恶意软件家族的关系和趋势给予洞察力。

3 设计

在本节中，我们将详细介绍DEEPREFLECT的管道，如 ，以及它使用的特征和模型。

3.1 概述

DEEPREFLECT的目标是识别恶意软件二进制中的恶意功能。在实践中，它通过定位异常的基本块（*感兴趣的区域* - RoI）来识别*可能是*恶意的功能。然后，分析人员必须确定这些功能是否表现出恶意或良性行为。我们的管道有两个主要步骤，如图2所示：（1）RoI检测和（2）RoI注释。RoI检测是通过一个自动编码器进行的，而注释则是通过对每个功能的所有RoI进行聚类并对这些聚类进行标记。

术语。首先，我们定义我们的 "恶意行为 "的含义。我们根据识别恶意软件源代码的核心组件（例如，拒绝服务功能、垃圾邮件功能、键盘记录器功能、命令和控制（C&C）功能、利用远程服务等）来生成我们的基础真相。这些都很容易被MITRE ATT&CK框架[9]所描述，该框架旨在将这些术语和行为的描述标准化。然而，当静态逆向工程我们的评估恶意软件二进制文件（即野生恶意软件二进制文件）时，我们有时不能肯定地将观察到的低级功能归于这些高级描述。例如，恶意软件可能会因为一些不同的原因而修改注册表键（其中许多可以由MITRE描述），但有时很难确定哪个注册表键因为什么原因被修改，因此只能粗略地标记为 "防御规避"：修改

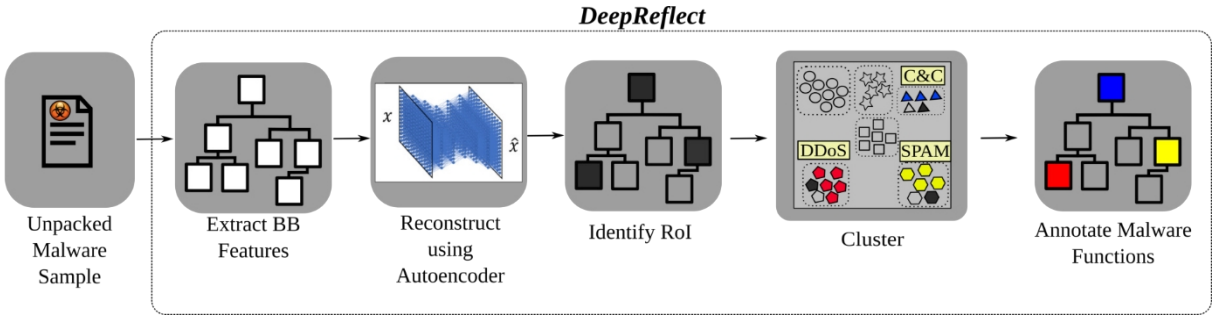

图2：DEEPREFLECT的概述。我们的系统将未打包的恶意软件样本作为输入，从每个输入（基本块（BB））中提取CFG特征，将其应用于预训练的自动编码器模型，以突出RoI（*感兴趣的区域*）。最后，它对这些区域进行聚类和标记。

MITRE的 "注册"。甚至像CAPA[3]这样的现代工具也能识别这些类型的模糊标签。因此在我们的评估中，我们将 "恶意行为 "表示为可以由MITRE框架描述的功能。

RoI检测。检测的目的是自动识别恶意软件二进制中的恶意区域。例如，我们希望检测C&C逻辑的位置，而不是检测该逻辑的具体组件（例如，网络API调用connect()、send()和recv()）。RoI检测的好处是，分析人员可以迅速指出负责启动和操作其恶意行为的具体代码区域。之前的工作只注重创建*临时*签名，仅根据API调用就能将二进制文件识别为恶意软件或某些功能。这对分析人员扩大工作范围特别有帮助（即不单单依靠人工逆向工程和领域专业知识）。RoI注释。注释的目的是自动标注包含RoI的函数的行为。换句话说，我们管道的这一部分确定了这个恶意功能正在做*什么*。使这种标记不影响分析员的工作流程并可扩展是至关重要的。分析师为标记集群所做的初始工作是一个长尾分布。也就是说，前期的工作相对较多，但随着他们继续给每个集群贴标签，工作会越来越少。这个过程的好处很简单：它可以给

分析师一种自动生成关于未见过的样本的报告和见解的方法。例如，如果一个恶意软件样本的变体包含与先前恶意软件样本类似的逻辑（但

对分析师来说，看起来足够不同，以至于不熟悉），我们的工具，让他们更快意识到这一点。

3.2 误差检测

恶意的概念/模式。

给定一个样本的重建$M(x)=x\hat{}$，通常通过计算均方误差(MSE)并检查所产生的 标量是否高于给定的阈值φ来识别恶意样本。MSE的计算方法是

$$MSE(x,x\hat{}) = \frac{1}{m}\sum x^{(i)} - x\hat{}^{(i)}{}^{2} \quad (1)$$

其中$x^{(i)}$是x中的*第*i*个*特征。

我们的假设是，与良性二进制文件相比，恶意软件二进制文件将包含类似但独特的功能。鉴于这种直觉，我们在一个代表各种行为和功能的不同的良性数据集上训练M。与之前将*整个*样本识别为恶意的工作相反，我们在 每个样本中识别恶意*区域*。具体来说，我们计算本地化的MSE，定义为

$$LMSE(x,x\hat{}) = x^{(i)} - x\hat{}^{(i)}{}^{2} \quad (2)$$

然后对得到的向量应用一个阈值φ来识别M没有识别或理解的模式。每个收到超过φ的平方误差的区块被称为感兴趣区域（RoI）。我们把在样本x中识别的RoI的映射集合表示为

$$R = \underset{x}{x^{(i)}}{}^{(i)} - x\hat{}^{(i)}{}^{2} \geq \varphi \quad (3)$$

自动编码器是一个神经网络M，它由一个编码器$En(x)$和一个解码器$De(e)$组成，前者将输入的x压缩成一个编码e，后者从一个给定的e重建x、

网络学习总结$x \in X$的分布，其中$X \subset R^m$。在诸如[43]的工作中，已经表明自动编码器可以检测恶意（异常）行为，当

在一个良性分布上训练出来的。这是因为M将无法重建x中的特征，因为M会召回

由R_x代表的亮点与监督分类器（如图像分类）的SHAP[40]前计划相似。然而，我们的方法是用来解释无监督的神经网络异常检测器（即在*未标记的*数据集上训练的），而SHAP是用于监督分类 模型（在*标记的*数据集上训练）。

3.2.1 特点

当给定一个二进制样本时，我们提取特征，将样本总结为x。有许多静态特征，在以前的工作中被用于恶意软件检测（例如，代码部分熵，导入的API调用等）[29，35，53，61，63]。然而，对于M来说，要在二进制文件中定位恶意行为、

我们的特征必须1对1地映射到原始样本中。因此，我们将每个二进制数据表示为一个*m-by-c*的矩阵，该矩阵用*c*个特征来概括前*m*个基本块的每个活动。一般来说，基本块是一系列的指令，以控制转移指令结束。当然，基本块的表示方法可能因反汇编程序而异，所以这个严格的定义可能不适用于所有静态恶意软件分析系统。我们的*c*特征是从先前的工作中发现的，即归属控制流图（ACFG）特征[23，75]。在这些作品中，ACFG特征被选择来执行二进制相似性，因为他们假设这些特征（由结构和数字CFG特征组成）将在多个平台和编译器中保持一致。虽然可以说我们的目标是相似的（即识别二进制的相似性和差异），但我们专门为研究恶意软件定制了这些特征。特别是，我们为自动编码器选择了我们的特征，以捕捉更高层次的行为。我们的特征包括每个基本块中指令类型的计数（ACFG特征中提取的更详细的形式）、CFG的结构特征和API调用的类别（已被用于来总结恶意软件的程序行为[18]）。

在DEEPREFLECT中，我们设置*m*为前20k个基本块。我们选择这一点是因为我们95%的数据集样本有20k个或更少的基本块。我们将*c*设定为18个特征，这些特征概括了每个基本块的情况：

结构特征。我们使用的结构特征是每个基本块的子代数和间性得分。这些特征可以代表一个控制流结构，通常用于网络通信（如连接、发送、接收）和文件加密（如查找文件、打开、读取、加密、写入、关闭）等操作。图6中可以找到一个实际的恶意软件样本中的这种功能的例子。

算术指令。我们使用的算术指令特征是每个基本块中包含的 "基本数学"、"逻辑运算 "和 "位移 "指令的数量。这些特征可以用来表示高级行为的数学运算是如何进行的。它们说明了数字是如何与功能互动的（例如，加密功能可能包括大量的xor指令，混淆功能可能包括逻辑和位移操作的组合，等等）。我们从英特尔架构的软件开发者手册[26]中检索到这些指令。此外，我们在图9中提供了一个恶意软件样本的例子，展示了这些类型的功能。

转移指令。我们使用的转移指令特征是每个基本块中 "堆栈操作"、"寄存器操作 "和 "端口操作 "指令的数量。这些特征可以用来表示高层行为的转移操作是如何进行的。它们说明了提供给函数的参数（以及

从函数调用返回的值）与该函数内的其他数据相互作用。它可以表明复杂的逻辑和数据操作（例如，去伪存真/解密可能会涉及更多的移动相关指令，C&C逻辑会涉及更多的堆栈相关指令，因为它调用更多的内部/外部函数）。我们同样从英特尔架构的软件开发者手册[26]中检索到这些指令。

API调用类别。我们使用的API调用特征是每个基本块中与 "文件系统"、"注册表"、"网络"、"DLL"、"对象"、"进程"、"服务"、"同步"、"系统信息 "和 "时间 "相关的API调用数量。这些分类的灵感来自于先前的恶意软件聚类工作[18]。这些特征可以用来表示执行恶意活动所需的高级别库操作，如网络通信和文件系统、注册表和进程操作。由于这些直接代表高层次的行为，它们对于理解一个函数的整体行为至关重要。图6和图8中可以看到利用这些不同的调用类型来执行不同行为的恶意软件功能的例子。

我们认为，这些特征比经典的ACFG特征更适合于恶意软件，因为(1)它们包括API调用，这些调用已在先前的工作中被用于恶意软件检测、(2)指令类别更细，允许对每个基本块有更多的上下文（如前所述），以及(3)它们不依赖于太容易被规避攻击的字符串[77]。当然，如果有一个有动机的对手，任何机器学习模型都可以被攻击，并被欺骗产生一个不正确的和非预期的输出。虽然我们的特征和模型并不是一个例外，但我们认为它们足以产生一个可靠的模型（即，它的行为符合预期），并使它足够困难，以至于对手不得不广泛地工作以产生一个误导性的输入（正如第4.7节所展示的）。关于针对我们系统的潜在攻击，请参考第5节。

3.2.2 模型

为了训练M，我们从各种良性的二进制文件中创建一个训练集X，其中$x \in X$是一个m乘c的特征向量，代表其中一个二进制文件。对于自动编码器的模型结构、我们使用一个U-Net[57]。U网已经被证明在生成性图像任务中表现良好，如生物医学图像分割和创建假图像。使用U-Net的好处是它在En和De之间有跳过的连接，M可以用它来跳过对某些特征的压缩，以保留$x\hat{}$中更高的保真度。

我们在X上训练M，目的是使重建损失最小化。该损失是输入和输出之间的共同L2损失，其定义为

$$L_2(x, x\hat{}) = \sum (x - x\hat{})^2$$

(4) 一旦训练完毕，M就会得到一个未见过的静态特征x。

恶意软件样本。然后，我们使用公式2突出潜在的恶意代码区域，这将在后面的第4节中进一步讨论，这样，任何超过该值的MSE被认为是一个RoI。在突出显示RoI（基本块）后，我们将它们所属的功能集中在 。

3.3 RoI注释

给定一个新的样本x，我们要确定其每个函数的行为（类别）。[1]并将其报告给莫莉。由于给*所有的*函数贴上标签是不现实的，我们只对*少数*函数进行注释，并使用聚类分析来传播结果。现在我们将解释这个过程是如何在接收Molly的样本之前设置的 。

3.3.1 聚类的特点

让x是一个从解压缩的恶意软件集合中提取的特征二进制。让F是使用BinaryNinja找到的x中的函数集合。对于每个$f_i \in F$，我们表示f中的RoIs$_i$作为q_i，其中$q_i \square R_x$。

我们创建一个用于聚类的训练集D，如下所示：给定恶意软件x_i，对于每个$q_i \neq \emptyset$，我们将f_i的行为总结为$\frac{1}{|q_i|}\sum_i q_i$，并将其添加到$D$中。这对所有恶意软件重复进行在我们的收藏中。

实验中，我们发现f_i's RoIs的这种表示方式最能体现函数在集群质量方面的行为（即使用Silhouette Coefficient & Davies Bouldin Score）。

3.3.2 聚类模型

为了对D中的函数进行聚类，我们首先将维度从18降到5，这样我们就可以扩展到50万个函数。降维是使用原理成分分析（PCA）进行的。接下来，我们使用HDBSCAN[6]对减少的向量进行聚类，并将D的聚类表示为C。HDBSCAN是基于密度的聚类算法DBSCAN的一个变种。我们选择HDBSCAN的原因是：（1）它可以识别非凸的聚类（与k-means不同）；（2）它可以自动选择聚类密度的最佳超参数（与经典聚类算法不同）。DBSCAN）。

3.4 部署

接下来，我们将描述DEEPREFLECT是如何被恶意软件分析者部署和使用的。

初始化。为了初始化DEEPREFLECT，Molly开始解压良性和恶意软件的二进制文件。然后，她把它们传递给DEEPREFLECT，DEEPREFLECT(1)提取我们的静态特征，(2)在良性样本上训练一个自动编码器模型M，(3)从每个恶意软件样本中提取RoIs R_x，(4)通过平均RoIs(q_i)总结每个函数的行为，作为D，和
(5)用PCA减少摘要，并将其聚类为C。

在这一点上，莫利现在已经确定了几组行为（功能），这些行为是恶意的（异常的），根据*M*.她现在可以对一小部分功能进行注释，或者继续进行她的常规工作，同时向*D* 添加注释（如前所述）。

执行。当Molly 收到一个新的样本*x 时*，这些行为被DEEPREFLECT 自动可视化、定位并为她贴上标签，具体如下：(1) 使用unipacker[11] 对x 进行解包，(2)*x* 通过*M*，获得RoIs R_x ，(3)使用BinaryNinja识别功能，通过平均RoIs将每个功能总结为*q*、(4) 使用PCA 模型减少剩余的函数摘要，(5) 将每个函数与与它最相似的聚类相关联、[2](6) 将大多数聚类注释分配给函数，并将结果映射到Molly的用户界面上。这个工作流程如图2所示。

然后，莫莉调查了突出显示的功能，在这样做的同时，她（1 ）对恶意软件正在做什么获得了更好的视角，（2）用相应的MITRE类别（动态更新*D*）注释任何标记为 " 未知 " 的功能，以及（3 ）能够观察到 其他恶意软件样本和家族之间的共享关系，通过其共享集群。

4 评价

在本节中，我们介绍了我们对DEEPREFLECT 的评估。首先，我们概述了每个评估实验的目标，并列出实验实现的研究目标（第2.4 节）。我们评估了DEEPREFLECT 的（1 ）可靠性，在我们编制的三个真实世界的恶意软件样本上运行，并将其与机器学习分类器、基于签名的解决方案和功能相似性工具进行比较；（2 ）凝聚力，让恶意软件分析师随机采样并标记野外样本中识别的功能，并比较DEEPREFLECT 如何将这些功能聚在一起、(3) 通过计算分析人员对整个恶意软件二进制文件进行逆向工程的功能数量来确定重点，(4) 通过观察共享相同功能的不同恶意软件家族以及DEEPREFLECT 如何处理新进入的恶意软件家族来确定洞察力，以及(5) 通过混淆和修改 恶意软件的源代码来试图逃避DEEPREFLECT。

4.1 数据集

构建一个良好的良性数据集对我们的模型的性能至关重要。如果我们不提供足够多的良性二进制文件的不同行为，那么恶意软件二进制文件中的一切都会显得不熟悉。例如，如果我们不在进行网络活动的二进制文件上训练自动编码器，那么*任何*网络行为*都*会被强调。

为了收集我们的良性数据集，我们在2018 年抓取了CNET[4]的便携式可执行程序（PE）和微软安装程序（MSI）。

[1]二进制中的函数是使用诸如BinaryNinja这样的工具在CFG上启发式地、静态地找到的。

[2]这可以通过测量中心点距离，使用增量DBSCAN，或通过重新聚类*D 来*实现（这就是我们在本文中所做的）。

类别	尺寸	类别	尺寸
驱动程序	6,123	商业软件	1,692
游戏	1,567	公用事业	1,453
教育	1,244	开发者工具	1,208
音频	1,023	安全问题	1,000
通讯	994	设计	844
数码照片	826	视频	787
定制化	778	生产力	730
桌面增强功能	699	互联网	695
联网	612	浏览器	440
首页	390	娱乐	257
帐户	43	旅行	17

表1：良性数据集：来自CNET的22个类别。

标签	紫外线	vobfus	赤铁矿	寿命	冰点
尺寸	3,438	3,272	2,349	1,313	914
标签	wapomi	寰球	pykspa	荃湾区	开始冲浪
尺寸	880	720	675	470	446

表2：恶意软件数据集：前10个人口最多的家族。

我们从CNET定义的22个不同类别中收集文件，以确保良性文件类型的多样性。我们总共收集了60,261个二进制文件。在给我们的数据集贴上标签后，我们通过Unipacker[11]来运行我们的样本，这是一个提取未打包的可执行文件的工具。虽然与先前的工作[21，58]相比，该工具并不完整，但如果它是成功的（即恶意软件样本是使用Unipacker设计的几种技术中的一种来解压的），就会产生一个有效的可执行文件。由于Unipacker涵盖了大多数恶意软件使用的流行打包器[67]，因此在我们的数据集上使用这个工具是合理的。默认情况下，如果Unipacker不能成功解压一个文件，它将不会产生输出。Uni- packer能够解压34,929个样本。然而，即使在解包之后，我们发现有一些样本似乎仍然是部分打包或不完整的（例如，缺少导入符号）。我们进一步过滤了没有有效起始地址和导入表大小为零的PE文件（也就是说，可能没有被正确解包）。我们还对解压后的二进制文件进行了重复计算。通过提取每个文件内容的SHA-256哈希值来确定其唯一性。为了提高数据集的质量，我们只接受被少于三家反病毒公司（根据VirusTotal）列为恶意的良性样本。在过滤之后，我们总共获得了23,307个独特的样本。每个类别的规模可以在表1中找到。

为了获得我们的恶意数据集，我们在2018年期间从VirusTotal[12]收集了64245个恶意软件PE文件。然后我们通过AVClass[62]运行这些样本，以检索恶意软件家族标签。与良性样本类似，我们对样本进行了解包、演绎和过滤。Unipacker能够解包47,878个样本。总的来说，我们留下了来自4407个家族的36396个独特的PE文件（其中3301个是单子家族--即只有一个样本属于该家族）。表2中列出了人口最多的前10个家族的规模。在收集了我们的数据集后，我们使用行业标准的二进制分解器BinaryNinja从每个样本中提取了我们的特征，并根据以下标准对每个特征向量排序

其基本块的地址在样本二进制中的位置。

4.2 模型设置

在提取了我们的数据集后，我们在80%的良性数据集上训练了自动编码器，并在剩下的20%上进行了测试。我们使用的核大小为24，跨度为1，并对特征向量进行了归一化处理；我们根据经验发现这些参数可以改善结果。我们对模型进行了最多10次的训练，得到的训练MSE为2.5090e-07，测试MSE为2.1575e-07-请注意，MSE值越低意味着良性样本的重建效果越好。在NVIDIA GeForce RTX 2080 Ti GPU上训练该模型大约花了40个小时。[3]

4.3 评价1 - 可靠性

为了评估DEEPREFLECT的可靠性，我们探索并对比了这些模型在二进制文件中定位恶意软件 ，的性能。

4.3.1 基线模型

为了评估DEEPREFLECT的自动编码器的定位能力，我们将其与一种用于定位样本中的概念的一般方法和特定领域的方法进行比较：(1) SHAP，一个分类模型解释工具[40]，(2) CAPA[3]，FireEye的一个基于签名的工具，用于识别二进制文件中的恶意行为、[4]和（3）FunctionSimSearch[5]，一个函数相似性工具。

给定一个训练有素的分类器和样本x，SHAP为x中的每个特征$x^{(i)}$，为分类器的预测提供一个贡献分数。对于SHAP的模型，我们训练了一个改良的深度神经网络VGG19[64]，以预测样本的恶意软件家族和样本是否是良性的。对于这个模型，我们不能使用我们的特征，因为该模型不会收敛。相反，我们使用了没有字符串或整数特征的经典ACFG特征。我们称这些特征为*属性基本块*（ABB）特征。我们对这个模型进行了分类训练（在恶意和良性样本上），取得了90.03%的训练精度和83.91%的测试精度。除了SHAP，我们还在ABB 特征上训练了另一个自动编码器，以便与我们在§3.2.1中解释的新特征进行比较。

4.3.2 地面真相数据集

对于我们的基础事实，我们静态地确定了三个不同恶意软件的源代码中的恶意组件（功能）的位置。我们通过匹配标记（如字符串和API调用）在二进制的CFG中找到这些功能，并将相应的基本块标记为恶意的。所有其他区块我们都标记为良性。我们注意到，我们无法定位14%到30%的恶意功能。

[3]为了保证可重复性，我们的源代码和数据集可以在以下网站找到 https://github.com/evandowning/deepreflect。

[4]我们使用了社区和专家规则集v1.2.0，来自于 https://github.com/fireeye/capa-rules

(取决于样本），所以它们被标记为良性。这些函数没有被发现的原因是：（1）由于二进制文件中的模糊和部分标识符（调用和字符串），这些函数无法被识别；（2）由于静态反汇编器对函数识别的限制，如动态解析函数和反静态分析技术[16]，这些函数被丢失。注意，被遗漏的函数在结果中反映为假阳性（FP）（图3），所以技术上我们的假阳性率（FPR）在现实中是比较好的。

构成我们地面真相的三个恶意软件样本是rbot、pegasus和carbanak。我们选择rbot是因为它是2004年的一个老的互联网中继聊天（IRC）僵尸网络，它仍然存在于常见的恶意软件饲料中--也就是说，它仍然出现在野外。我们选择它还因为它编译成一个PE文件（可直接与我们数据集中的PE恶意软件样本相比较）。我们选择pegasus是因为它是2016年的一个较新的银行木马，由多个有效载荷（PE文件和DLL文件）组成。这使我们能够在可能在内存或其他地方捕获的文件上评估我们的工具（即不只是假设所有的恶意软件会整齐地将其所有的行为打包到一个文件中）。最后，我们选择carbanak是因为它是最近泄露的2014年的银行恶意软件，使它仍然相对现代。行为、代码布局和实现以及恶意软件家族类型和年龄的多样性是我们选择这三个样本的原因。

4.3.3 结果

这个实验的结果可以在图3中找到。为了得到每个函数的值，我们将其相应的基本块SHAP（将负值设置为0）或MSE值相加。DEEPREFLECT与SHAP。SHAP的目标是识别模型输入中影响模型分类决策的区域。虽然单独的恶意软件分类器为分析师提供了输入的恶意软件家族，但SHAP将识别输入中最重要的区域，以做出该决定。因此，从概念上讲，它可以用来识别不同的恶意软件家族和良性软件之间的差异（如前所述）。然而，这可能并不完全有效。每当发现一类新的恶意软件时，分析人员就必须不断地重新训练模型，而且由于SHAP的递归算法（在神经网络中来回多次传递），它本身就很慢。DEEPREFLECT通过利用无监督学习和只需要通过一次神经网络来检索模型的输出，克服了这些问题。

DEEPREFLECT与CAPA。接下来，我们将DEEPREFLECT与CAPA[3]进行了比较，CAPA是一种静态识别可执行文件中能力的工具。它通过使用描述各种行为的手写签名来完成这一工作。例如，"连接到HTTP服务器"、"创建进程"、"写文件 "等。由于CAPA是基于签名的，它有可能因为缺乏通用性而错过恶意行为，而DEEPREFLECT是使用无监督学习进行训练的，没有这种限制。对于DEEPREFLECT、

我们选择检测阈值φ的方法如下：首先，我们绘制了所有地面实证样本的ROC曲线（图3）。然后，我们为每个样本确定了单独的阈值，该阈值达到了80%的真实阳性率（TPR）。我们选择这个TPR是因为它足够大，可以检测到大多数的恶意功能，同时保持相对较低的FP（用于审查单个样本）。

CAPA未能识别行为的一个例子是当API调用符号被恶意软件混淆时（例如，在运行期间动态解析API调用的名称）。因此，它错过了KeyLoggerThread()函数，该函数调用各种动态解析的API调用来记录受害者的击键。但由于这里没有有趣的API调用，CAPA错过了它。DEEPREFLECT能够成功地识别它，因为它*不*完全依赖API调用和签名来发现恶意行为。

DEEPREFLECT无法识别CAPA（据说）所做的行为的一个例子是一个内部功能，它将发送的文件传输到C&C服务器。DEEPREFLECT本应在概念上发现这一点，但它未能做到这一点。然而，API调用都是模糊的，所以CAPA在这里应该是失败的。经过进一步调查，CAPA认为这里有一个检索文件大小的调用，尽管在源代码中并不存在这样的调用。检查一个邻近的函数，我们发现它调用了GetFileSize()。因此，我们认为这是CAPA的默认反汇编程序和BinaryNinja之间反汇编函数地址不一致的一个例子（因为两者可能使用不同的方法来检测函数边界）。在这种情况下，DEEPREFLECT发现了CAPA的所有恶意功能。虽然我们的工具没有成功捕捉到上述的恶意函数（由于我们设置的阈值），但它仍然比基于签名的工具（如CAPA）更通用和可扩展，因为后者依赖于API调用和字符串。

DEEPREFLECT vs FunctionSimSearch。FunctionSimSearch是一个由Google Project Zero开发的函数相似性工具[5]。我们用默认参数对我们的数据集中的良性函数进行了数据库训练。在对良性数据集进行训练后，我们用地面真实数据集中的函数对其进行查询。我们指定该工具输出前1000个最常见的函数和它们的相似度分数。我们之所以选择1000个，是因为该工具返回查询的速度（1小时）和插入数据库的函数数量（1,065,331个函数）。如果把它作为一个异常检测器，我们会期望不熟悉的函数（即恶意函数）会导致明显小于熟悉的函数的相似度得分。从图3中可以看出，它的表现很差。应该指出的是，对性能差的一个可能的解释是由于函数边界之间的分歧（不同的反汇编工具是常见的），但这不应该是急剧的差异（如看到CAPA的反汇编工具表现更好）。

来自野外的样本。用于验证DEEPREFLECT的

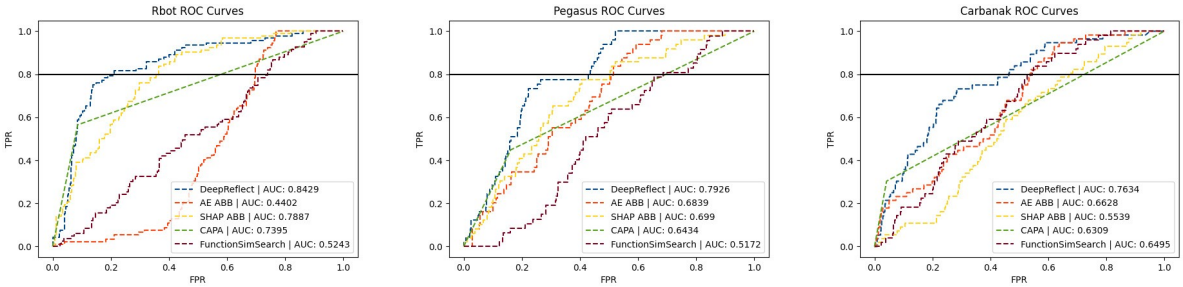

图3：DEEPREFLECT、使用ABB特征的AE、使用ABB特征的SHAP、CAPA和FunctionSimSearch在三个地面真实恶意软件样本上的ROC图（每个阈值的性能）。水平黑条代表TPR为80%。

为了提高识别野生样本的恶意功能的能力，我们从virut家族中随机选择了一个二进制文件。我们选择这个二进制样本进行逆向工程，因为它很简单（即，它相对较小，能够逆向工程每一个内部功能），而且virut在2006-2013年及以后一直是一个被充分研究的僵尸网络。首先，一个恶意软件分析师使用CAPA和BinaryNinja对这个样本进行逆向工程，手动检查所有39个内部功能，并将它们标记为恶意（根据MITRE框架）或良性。接下来，分析员在这个样本上执行了DEEPREFLECT，它识别了15个RoI。将其与我们分析师的手工分析相比较，我们最初认为DEEPREFLECT漏掉了一个功能（比较一个参数的逻辑，这将导致终止恶意软件的处理器或不终止）。由于CAPA的默认反汇编程序的反汇编和BinaryNinja的反汇编之间的差异，功能地址（边界）是不相同的。在这种情况下，CAPA在这个内部函数上识别了进程终止，而BinaryNinja在这个函数位置上不包含这种逻辑。由于这种差异，DEEPREFLECT基本上捕捉到了所有恶意的功能。此外，我们让一个分析师在他过去分析过的恶意软件上使用DEEPREFLECT。这将在§A.1中详细讨论。

摘要。我们已经表明，我们在DEEPREFLECT中的自动编码器定位方法实现了目标G1和G3，无需对恶意软件样本或标记数据进行训练即可识别二进制文件中的恶意行为。此外，我们还证明了它比一个流行的解释框架（SHAP）和基于签名的方法（CAPA）要好。最重要的是，DEEPREFLECT比SHAP（速度较慢，需要标注数据集）和CAPA（基于签名的解决方案）更实用，因为该模型不需要让专家对恶意软件或其组件进行昂贵的标注过程。最后，我们已经表明，我们的 特征比ABB特征表现得更好。

4.4 评价2 - 凝聚力

为了评估DEEPREFLECT对AE识别的恶意软件组件进行分类的能力，我们在五个 经验丰富的恶意软件分析师的帮助下，探索半监督聚类模型的质量。

4.4.1 实验设置

首先，我们使用自动编码器M，在25206个恶意软件样本中识别出593181个恶意组件（功能）。这比原来的~36000个样本要少，因为有些样本要么（1）从未完成提取特征，（2）没有检测到超过选定阈值的RoI，或者（3）RoI不存在于二进制中--其结果令我们困惑，但可以解释为数据损坏，在提取特征和运行聚类之间自动升级BinaryNinja的一些问题，或者因为基本块存在于我们不考虑的函数（即外部函数）中）。

为了对大量的恶意软件样本函数进行聚类，我们希望将FPR保持在5%的低水平。在工业和现实世界环境中，较低的FPR往往比TPR更有价值。使用这个阈值（在我们的真实样本上产生了40%/5%的综合TPR/FPR），我们使用DEEPREFLECT来提取和聚类已识别的函数为C（§3.3）。在对函数特征向量运行PCA后，HDBSCAN使用默认的超参数产生了22469个聚类。最大的集群包含6,321个函数，最小的集群包含5个。

在图10中，我们展示了聚类规模的分布。该图显示，存在一个长尾分布（在基于密度的聚类中很常见），前10个最多的聚类占到了5%的功能。
反向工程师们。为了评估聚类质量，我们招募了五名具有2-7年逆向工程经验的恶意软件分析员。

五位分析员随机抽查了一些功能，并使用MITRE ATT&CK[9]的分类法对其进行了标注。如果这些功能被认为是良性的，分析员就给它们贴上这样的标签。总的来说，分析员从25个最大的恶意软件家族RoI（因其规模和行为的多样性而被选中）中随机抽取了177个功能（用于176种不同类型的MITRE ATT&CK标签）。时间是选择多少功能的一个限制因素。虽然177个功能与提取的60万个功能相比很小，但对每个功能进行反向工程需要15-30分钟（有时更长）。然后，我们选择了一位分析员对这些功能进行手工分组。最后，我们将手工分组与DEEPREFLECT的分组进行了不同的比较。

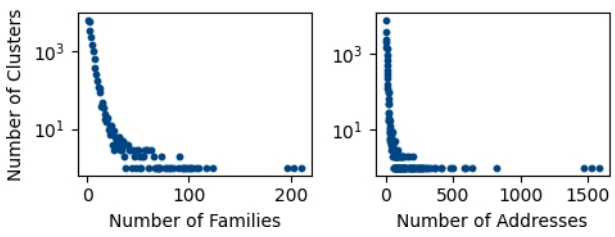

图4：聚类的多样性。左图：每个集群的家族分布。右图：每个群组的地址分布，显示功能位置没有偏差。

测量。在表3中，我们列出了分析员在工作中遇到的各种MITRE ATT&CK标签。

4.4.2 结果 - 集群质量

在对函数进行手工标注后，分析家们最终得到了78个恶意的集群。有5种情况，手工制作的集群出现在*C*的*不同*集群中。为了简洁起见，我们在这里只讨论三种情况。在第一种情况下，位于手工集群中的两个函数被分析员认为是相似的。它们都是调用SetEvent()的小函数，但内容不尽相同。其中一个函数多了一条指令，将子程序参数的值+0x40偏移量设置为0。这对分析员来说并没有足够的区别，所以他们将它们归类为相同的。然而，对HDBSCAN来说，特征向量的内容会发生变化，因此（取决于参数）可能将这两个函数分开。这是一个HDBSCAN过于敏感的案例。在第二种情况下，有三个函数被认为与分析师相似，但在*C语言*中被分成两个集群。不同的函数包含一个前提条件IsProcessorFeaturePresent()，然而都在GetCurrentProcess()上调用TerminateProcess() - 因此它们的行为足够接近，可以将它们标记为 "Discov- ery：虚拟化/沙盒规避"。这些都是沙盒规避的表现，因为这些技术寻找的是沙盒中的进程与真实主机上的进程之间的差异[54]。通常情况下，恶意软件试图退出的唯一原因之一是他们从C&C服务器收到了这样的命令，或者他们处于一个不受欢迎的环境（不适合恶意软件感染或被确定为分析环境）。在第三种情况下，一个手工制作的集群包含两个用*C语言*分开的函数，与其他情况类似，这些函数的内容足够接近，但不完全相同。它们都以同样的方式执行GetTickCount()以及调用其他各种内部函数。有8个案例，手工制作的集群被合并到*C语言*中的同一个集群中。

虽然出现了这些错误，但89.7%的分析师手工制作的聚类函数与我们的工具所创造的相匹配。因此，我们认为聚类的结果是值得信赖的。在未来，HDBSCAN的参数可以被调整以纠正这些差异。

误差边际。我们现在评估有多大比例的集群是良性的还是恶意的。在给随机抽样的函数贴标签时，我们看的是具有 *一致*标签的手簇。有时，我们的分析员在给一个函数贴上什么MITRE标签的问题上彼此意见不一。为了保持一致性，我们只考虑那些分析师们同意的标签。我们发现，在119个功能中，60.5%是恶意的，39.4%是良性的，误差率为9.29%。检查所有函数的百分比（不管它们的集群），我们发现类似的百分比结果。请注意，在第4.3.3节中，我们的地面真实样本的假阳性率要低得多。这是因为他们只从最大的恶意软件家族RoI中选择（即对整个60万人口不是统一随机的）。这样做是为了确保分析人员审查最常见的提取功能，这使分析人员有更好的机会发现共同分享的恶意功能，如C＆C行为，反分析行为，等等。

摘要。恶意软件分析师发现，DEEPREFLECT的集群是一致的（无论恶意软件家族或函数在二进制中的位置）。虽然所选样本的数量应能捕捉到群体，但在更大的样本量上，结果可能会有所不同。我们还发现，该聚类与分析师的 手动聚类的函数的89.7%相匹配，有助于实现目标G1。

4.5 评价3--重点

从以前的工作[69]和与其他分析师的讨论中，我们发现，恶意软件分析师的静态逆向工程工作流程首先是对恶意软件二进制文件中各种功能的 *位置*形成假说。这通常是通过观察可疑的字符串（如URL，域名）或API调用（如连接或发送）的位置来完成的。然而，正如第4.3节中所展示的，这些指标不能单独依赖。DEEPREFLECT的好处是它能够集中恶意软件分析师的注意力，而不是让他们盲目地在每个二进制文件中搜索功能。我们通过以下方式进行评估：（1）计算每个恶意软件二进制文件的所有功能中高亮功能的百分比；（2）分析假阳性和DEEPREFLECT的潜在排名方案，以确定分析师应首先查看哪些高亮功能；以及（3）讨论假阴性和如何在未来减轻它们。减少工作量。对于每个恶意软件样本，我们提取了包含自动编码器发现的至少一个RoI的每个函数，并将其与二进制内部函数的总数进行比较。从图5中可以看出，大部分突出显示的功能使分析员要查看的功能数量减少了至少90%。最小的减少量是0%（即所有函数都被突出显示），最大的减少量超过99.9999%，平均减少量为85%。如果恶意软件样本中的功能数量较少，这些百分比本身可能会产生误导。就原始数字而言，每个恶意软件样本的最小/最大/平均 *高亮*功能数为1/527/23.53。

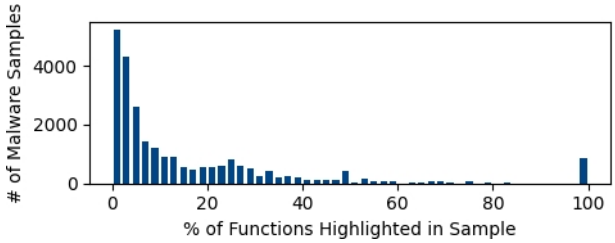

图5：功能计数。分析师必须审查的功能（每个恶意软件样本）的百分比。

分别是。每个恶意软件样本的最低/最高/平均*总*功能数分别为1/26,671/663.81。这表明，平均而言，分析人员只需审查24个功能，而不是664个功能。然而，我们需要进一步深入研究，因为这些功能可能是小规模的，因此很可能是微不足道的反向工程。

为了回答这个问题，我们计算了每个恶意软件样本中每个功能的基本块的数量。基本块可以作为一个函数的复杂性的指标。每个*突出的*功能中基本块的最小/最大/平均数分别为1/134,734/96.02。每个函数内基本块的最小/最大/平均数分别为1/22,947/16.51。这表明大多数突出显示的函数与平均函数相比要复杂得多，如果这些函数被自动标注给分析员，将会大大减少他们的工作量。

假阳性和优先级的确定

所有的安全解决方案都存在误报现象。对于那些在真实世界环境中工作的人来说，减少假阳性是一项永无止境的任务。当使用我们的集群阈值在我们的地面真实样本上运行DEEPREFLECT时，Rbot包含39个真阳性（TP）和23个FP，Pegasus包含22个TP和80个FP，而Carbanak包含8个TP和69个FP。虽然TPs相对较小（40% TPR），但FPs相对较小（5% vs 25% FPR）。为了进一步减少FPs，一个解决方案是根据MSE对DEEPREFLECT识别的函数进行分类（类似于我们确定聚类的阈值）。直观地说，MSE越高，函数就应该越可靠。当检查排名前100位的组件时，DEEPREFLECT/SHAP在rbot上的精度为0.629/0.487，在pegasus上为0.229/0.138，而在carbanak上为0.111/0.01。正如预期的那样，当添加更多的顶级组件时，精度会下降，因为模型对MSE较低的组件的信心较低（pegasus和carbanak都有较大的代码库，因为它们是更现代的恶意软件）。这些结果也与我们的分析师在A.1节中对Mikey的实践评估相一致，在那里，假阳性被置于MSE的倒数第三位。

按MSE值对函数进行排序并不总是可靠的。在这种情况下，可以利用其他的基本缓解策略。例如，分析员可以对DEEPREFLECT提取的函数使用简单的启发式方法（如CAPA中使用的方法），以

了解它可能属于什么行为类别。他们还可以根据功能与数据集中其他功能的独特性来确定其优先级，找到哪些功能是分析师以前没有见过的潜在的新出现的恶意行为。例如，按其相关群组的大小对功能进行排序（较小的群组表示更独特和不常见的功能）。

假阴性。假阴性也是所有安全解决方案中的常见现象。未知的威胁总是存在的，它们会躲避这些系统。使用相同的集群阈值，DeepReflect对rbot有53个假阴性（325个TN），对pegasus有27个假阴性（407个TN），对carbanak有48个假阴性（2,111个TN）。接下来，我们讨论来自我们的地面实证样本rbot的三个FN案例。第一个是一个函数CaptureVideo()，它对受害者的电脑进行增量截图。这个函数有许多对外部API的调用，这些API被混淆了（正如恶意软件通常所做的）。虽然我们证明了我们的工具能够捕获含有混淆的API调用的恶意函数，但它在这样做时并不总是可靠的，任何不能访问更高级别的函数调用的工具都会因此受到影响。第二个是函数getcdkeys()，它从受害者的主机中收集视频游戏安装密钥，并将其发送到攻击者的C&C服务器。再次，调用了混淆的注册表密钥API调用，这提供了关键的上下文信息。也可能是这样，一些良性的软件游戏执行这种完全相同的功能，以检查用户是否安装了视频游戏的有效副本。这说明了仔细采购训练数据集的必要性（如后面第5节所述）。最后，第三个FN是一个函数DDOSAttack()，它调用函数ResolveAddress()、SpoofIP()和SendDDoS()，发起攻击。这个函数可能被忽略了，因为它更像是一个启动恶意行为的调用函数。然而，这个调用函数提供了关于如何发起攻击的重要背景信息。为了减轻这种情况，未来可以使用一个简单的 "连带责任 "启发式方法，将调用可疑行为的函数识别为可疑的。此外，阈值可以根据分析者的目标来调整，即是增加TP还是减少FP。

最后，我们在附录A中详细介绍了由DEEPREFLECT确定的恶意功能的具体例子（并通过MITRE标记）。在那里，我们说明了一些行为，如用于文件丢弃的C&C通信（图6），文件和数据的去伪存真/解码（图7），以及搜索各种文件以复制其内容（图8）。

摘要。我们已经证明，DEEPREFLECT有能力将分析师的注意力集中在恶意软件样本的各种恶意活动上。对于大多数样本来说，它可以将其搜索空间平均减少90%和85%。这对于分析员需要对可能存在*的*恶意行为有一个高层次的了解时很有帮助，这样他们就可以更深入地分析（如调试）。这就满足了目标G2。

4.6 评价4--洞察力

为了评估DEEPREFLECT是否对恶意软件家族及其行为的关系提供了有意义的见解，我们探索了集群的多样性。图4的左侧绘制了C中每个集群的不同家族的数量，可以看出家族之间有许多共享的恶意软件技术和变体。

多样性。当然，大多数集群只有一个恶意软件家族（由图10所示的集群的长尾分布解释）。然而，10到1000个集群包括各种家族 - 有些甚至包含超过200个不同的家族。例如，tiggre和zpevdo家族共享一个 "执行：命令和脚本解释器 "的行为，它们调用GetCommandLineA()并解析所涉及的字符（如MITRE所描述）。单子样本。这些是只有一个样本的恶意软件家族。由于我们使用自动编码器，我们可以从单子样本中捕获新的行为。为了检查DEEPREFLECT是否能识别单子样本中的恶意功能，我们观察了我们的数据集中是否有单子样本与其他恶意软件家族聚集在一起。事实上，我们发现DEEPREFLECT识别了1,763个集群，其中至少包含一个单子样本。

新的恶意软件家族。接下来，我们研究当新的家族被引入到DEEPREFLECT时，会发生什么。我们对所有的恶意软件做了一个聚类模型C_1，除了四个家族。然后，我们将这些家族加入到该集合中，并将该集合聚类为C_2。当我们将C_1与C_2进行比较时，我们发现：(1)通过引入新的家族创建了新的聚类，(2)这些家族的部分功能被添加到旧的聚类中（即分析员将收到关于新家族的分类信息）。更多细节，见A.3节。总结。我们发现，DEEPREFLECT提供了对恶意软件行为关系的洞察力（G4）。在部署中，这种元信息可以与确定的，为分析人员提供即时的洞察力。

4.7 评价5--稳健性

混淆。鉴于对抗性机器学习的兴起，我们必须意识到，对抗者可能试图混淆他们的代码，以减轻DEEPREFLECT的生产力。因此，我们对DEEPREFLECT进行了评估，以应对混淆攻击的情况。我们没有对包装或密码器进行评估，因为这些超出了我们工具的范围。相反，我们利用Obfuscator-LLVM [31]（表示为*ollvm*）。我们利用*llvm*对rbot样本的源代码进行混淆，使用了五种技术：（A）控制流扁平化，（B）指令替换，（C）假控制流，（D）组合技术。(A）和（B），以及（E）结合技术（B）和（C）。检查提取和聚类的功能、DEEPREFLECT大多没有受到混淆视听的影响。这是有道理的，因为自动编码器突出了它不能识别的功能，而我们的特征包含API调用。

(这些都没有被*ollvm*修改)。详见§A.4。类似模仿的攻击。接下来，我们进行了一个简单的模仿攻击，我们将直接操纵我们的特征的良性代码插入到我们的真实样本的恶意函数中。所选择的良性代码来自一个开源的基本代码库，用于执行整数、字符串和文件I/O操作[10]。之所以选择它，是因为它已经被用作测试抗干扰能力的基准[10]。特别是，我们观察了使用DEEPREFLECT时每个函数的MSE值与我们在ABB特征上训练的AE相比有多大变化。我们针对12个函数（每个基础样本4个），这些函数来自各种行为（例如，反AV、键盘记录器、dropper、DDoS，等等）。对每个样本使用图3中TPR 80%的阈值，我们发现，与其他AE相比，DEEPREFLECT对这些修改后的函数（包括原始函数）输出的MSE值明显较大（几个数量级）。这表明，DEEPREFLECT更有信心将这些函数标记为恶意的。虽然这些攻击都没有能够持续地逃避这两个模型，但我们观察到，DEEPREFLECT的MSE值并没有急剧变化到足以引起关注。此外，我们观察到，有时将函数与文件I/O操作一起插入，导致DEEPREFLECT认为一个函数比它最初考虑的*更*异常（与在ABB特征上训练的AE相比，更加异常--这体现在尝试模仿攻击后，两个平均MSE值都增加了）。这也说明了攻击者所面临的困难：不是任何良性代码都可以插入到恶意函数中来逃避的。

为了增加绕过DEEPREFLECT的可能性，我们测试了两个更良性的功能：（1）在dropper恶意功能中加入微软网站托管的网络连接/发送实例，（2）在DDoS行为中加入同样的实例，以及（3）在远程代码执行中加入进程I/O创建实例，其中恶意软件启动一个'cmd.exe'进程。观察到同样的结果，除了DEEPREFLECT之外，我们的特征优于ABB的特征，认为它们更不熟悉。

摘要。尽管DEEPREFLECT没有受到*llvm*的混淆方法或我们的基本模仿实验的显著影响，但我们确信DEEPREFLECT可以被逃避。然而，这些实验表明，它不容易被这些基本攻击。

5 讨论

总而言之，我们证明了DEEPREFLECT可以重新识别恶意软件样本中的恶意活动（如§4.3和§4.6所示），这满足了我们的研究目标§2.4中的G1。通过其他实验，我们证明了该系统可以集中分析人员的注意力，处理新的恶意软件家族（如§4.4和§4.5所示），这满足了目标G2和G3。它还表明，DEEPREFLECT是

能够确定对恶意软件行为的共同功能的洞察力，满足G4（其余目标）。我们还表明，我们的工具比其他基线方法，如 可解释机器学习或基于签名的解决方案更好。

5.1 限制条件

每个系统都有弱点，我们的系统也不例外。对抗性攻击。一个有动机的对手可以在训练数据集中下毒[46, 51]，使自动编码器创造出
一个脆弱的模型，将有效地隐藏恶意软件的
功能。它们也可以混合在一起，看起来像一个良性的二进制[24, 70]。许多论文都探讨了在架构层面上攻击机器学习模型[47, 48, 73]。他们也可以在用于聚类的数据集上下毒[19]。虽然这些攻击确实存在，但常见的反措施[49, 65, 71, 74]可以在未来应用来颠覆它们。

对手也可以通过操纵这些特征来攻击我们的系统。然而，这可能被证明是困难的，因为我们的特征是基于不容易改变的特性。他们必须知道如何精确地修改CFG的结构、指令类型和使用的API调用类型，而不破坏恶意软件的*动态功能*。这不是简单的事情，无论是在编译前还是编译后。

训练数据质量。最后，我们的自动编码器模型在很大程度上取决于良性数据集的内容和质量。如果一些功能被排除在训练集之外，那么结果就会变得有偏差。例如，如果我们不包括任何执行网络行为的程序，那么看到的每一个网络行为都会被认为是恶意的。因此，我们必须谨慎选择各种良性软件来补充恶意行为。另一方面，如果我们对太多类似恶意的功能进行训练，我们的系统可能会在恶意软件中漏掉这些功能。例如，如果远程桌面协议（RDP）行为是我们良性数据集中的一个应用，我们的系统可能不会将任何RDP功能标记为恶意的。需要取得一个适当的平衡，使我们的系统能够检测到分析人员感兴趣的恶意功能。

人为错误。DEEPREFLECT在很大程度上依赖于人类分析员的经验和协议。在一开始，对pegasus地表水的标注就有问题--我们在最初的源代码标注中并不完美。经过调试，我们意识到，有一个函数通过远程桌面协议（RDP）删除了互联网连接的历史，这实际上*不是*一个FP。另一个所谓的FP催生了一个线程，与远程受害者的服务控制管理器（SCM）进行交互，这当然是一种恶意的行为。因此，我们需要更新我们的标签，因为还有其他这样的例子。虽然这最初可能看起来是一个限制，但我们认为这是一个潜在的教学应用。也就是说，有经验的分析师可以使用我们的工具，从恶意软件 样本中提供有标签的函数和代码的例子，以方便培训新的或经验不足的分析师。

6 相关作品

深度学习和恶意软件。最近，深度学习已经被恶意软件分析社区采用。大多数的目标是使用深度学习神经网络对恶意软件样本进行分类或检测[50, 66, 68]。Malconv[53]从可执行文件中提取原始字节值并在卷积神经网络（CNN）上进行训练。Neurlux[30]从动态沙箱报告中提取特征。甚至微软也举办了一个Kaggle比赛[8，56]，目标是采取二进制文件（没有附加PE头），并根据9个恶意软件家族对其进行准确分类。

使用静态和动态特征也对二进制相似性进行了研究[2,17,22,75]。虽然二进制相似性是一个与我们相似的问题，但它在一个重要的方面有所不同：他们的目标是将每个二进制文件与其他所有二进制文件进行比较，而我们将一个特定类型的二进制文件看起来像什么（良性二进制文件）编入CNN，并利用重建错误来告诉我们它不能识别哪些部分。我们的目标不是正式确定二进制文件之间的相似性--尽管我们确实扩展了我们的分析，以确定恶意软件家族之间的共同概念。

自动编码器和安全。本文并不是第一个在网络安全数据集上研究自动编码器的文章。[34]使用深度自动编码器来概括恶意软件样本的样子，并将结果提供给生成式对抗网络（GAN），试图挫败混淆恶意软件的静态技术（例如，重新排序功能布局）。其他论文[20，28，32，76]使用自动编码器生成输入来训练其他恶意软件分类器，以此来提高泛化能力。我们的工作有很大不同，因为我们在良性二进制文件上训练自动编码器，试图归纳出看起来正常的东西，并使用重构MSE来识别恶意软件二进制文件中的恶意功能。在[43]中，作者通过检测异常特征向量（网络流量统计的快照），使用自动编码器的集合作为NIDS。然而，[43]使用方程1来识别整个观察的异常，而DEEPREFLECT使用方程2在观察中定位一个或多个异常的自动编码器。

据我们所知，目前还没有任何相关工作利用机器学习对恶意软件中的恶意功能进行静态识别和定位，更不用说使用自动编码器的无监督方法 。

7 总结

在本文中，我们介绍了DEEPREFLECT：一个用于定位和识别恶意软件二进制文件中恶意组件的工具。该工具是实用的，因为它不需要标签数据集进行定位，只需要少量的标签进行分类--在分析员的常规工作流程中逐步收集。我们希望这个工具和公布的代码能够帮助世界各地的分析人员识别恶意软件样本中*的*，以及存在*哪些*恶意功能。

8 鸣谢

我们感谢匿名审稿人提供的有益和有价值的反馈。本资料部分得到了美国海军研究办公室（ONR）的资助，包括N00014-17-1-2895、N00014-15-1-2162和N00014-18-1-2662，和
美国国防部高级研究计划局（DARPA）根据合同HR00112090031。本材料中表达的任何意见、发现、结论或建议都是作者的观点，不一定反映ONR或DARPA的观点。

参考文献

[1] Binaryninja. https://binary.ninja/.

[2] Bindiff. https://www.zynamics.com/bindiff.html。

[3] Capa. https://github.com/fireeye/capa.

[4] Cnet。 https://download.cnet.com/windows。

[5] Functionsimsearch. https://github.com/googleprojectzero/functionsimsearch。

[6] Hdbscan 。 https://github.com/scikit-learn-contrib/hdbscan。

[7] Ida pro. https://www.hex-rays.com/products/ida/.

[8] 微软恶意软件分类挑战（2015年大）。 https://www.kaggle.com/c/malware- 分类。

[9] Mitre att&ck. https://attack.mitre.org/versions/v7/matrices/enterprise/windows/。

[10] Obfuscation benchmarks。 https://github.com/tum-i4/obfuscation-benchmarks。

[11] unipacker： https://github.com/unipacker/unipacker. 基于仿真的自动和独立于平台的windows二进制文件解包器。

[12] Virustotal。https://www.virustotal.com/。

[13] Virustotal statistics. https://www.virustotal.com/cs/statistics/, Aug 2020.

[14] 海勒姆-S。Anderson和Phil Roth。EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *arXiv preprint arXiv:1804.04637*, April 2018.

[15] Dennis Andriesse, Asia Slowinska, and Herbert Bos.二进制文件中的编译器预知功能检测。在*IEEE欧洲安全与隐私研讨会*上，2017年。

[16] 迈克尔-贝利Carbanak周第一部分：https://www.fireeye.com/blog/threat-research/2019/04/carbanak-week-part-one。a-rare-occurrence.html，2019年4月。

[17] Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna.高效检测恶意软件中的分裂个性。在*NDSS*，2010年。

[18] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 可扩展的、基于行为的恶意软件集群。在*NDSS*，第9卷，第8-11页。Citeseer, 2009.

[19] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli.中毒行为的恶意软件聚类。In *Pro- ceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 27-36.ACM，2014。

[20] Omid E. David和Nathan S. Netanyahu.Deepsign：用于自动恶意软件签名生成和分类的深度学习。在*2015年国际神经网络联合会议（IJCNN）上*，第1-8页。IEEE，2015年。

[21] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee.以太：通过硬件虚拟化扩展的恶意软件分析。在*ACM计算机和通信安全会议上*，第51-62页。ACM，2008年。

[22] Mohammad Reza Farhadi, Benjamin CM Fung, Philippe Charland, and Mourad Debbabi.Binclone：检测恶意软件中的代码克隆。在*2014年第八届软件安全与可靠性国际会议（SERE）上*，第78-87页。IEEE，2014年。

[23] 冯倩，周润东，徐成成，程瑶，Brian Testa，和尹恒。基于可扩展图形的固件图像错误搜索。在*2016年ACM SIGSAC计算机和通信安全会议上*，第480-491页。ACM，2016。

[24] Prahlad Fogla, Monirul I. Sharif, Roberto Perdisci, Oleg M. Kolesnikov, and Wenke Lee.多态混合攻击。在*USENIX安全研讨会上*，2006年。

[25] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, and others.BotMiner：对网络流量进行聚类分析，进行与协议和结构无关的僵尸网络检测。在*USENIX安全研讨会上*，第139-154页，2008年。

[26] 部分指南。Intel® 64和ia-32架构的软件开发者手册。*合并卷：1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, 和 4*, 2020.

[27] 尤塔姆-古特曼停止流失，避免倦怠：如何保持你的网络安全人员。*SentinelOne*, 2020年3月。

[28] William Hardy, Lingwei Chen, Shifu Hou, Ye Yanfang, and Xin Li.DL4MD：一个用于智能恶意软件检测的深度学习框架。在*国际数据科学会议（ICDATA）论文集*，第61页。世界计算机科学、计算机工程和应用计算大会（WorldComp）指导委员会，2016。

[29] Suman Jana和Vitaly Shmatikov。在恶意软件检测器中滥用文件处理，以获取乐趣和利益。在*IEEE安全与隐私问题研讨会上*，第80-94页。IEEE，2012年。

[30] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna.Neurlux：无需特征设计的动态恶意软件分析。在*第35届计算机安全应用年会论文集中*，第444-455页，2019年。

[31] Pascal Junod, Julien Rinaldini, Johan Wehrli, and Julie Michielin.Obfuscator-LLVM - 大众的软件保护。*在IEEE/ACM第一届软件保护国家间研讨会论文集，SPRO'15，意大利佛罗伦萨，2015年5月19日*，第3-9页。IEEE，2015年。

[32] Temesguen Messay Kebede, Ouboti Djaneye-Boundjou, Barath Narayanan Narayanan, Anca Ralescu, and David Kapp.使用基于自动编码器的深度学习架构对恶意软件程序进行分类，并将其应用于微软恶意软件分类挑战（大2015）数据集。In *2017 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 70-75.IEEE，2017。

[33] Hyang-Ah Kim和Brad Karp。Autograph：争取实现自动化、分布式的蠕虫签名检测。在*USENIX安全研讨会上*，第286卷。圣地亚哥，加利福尼亚州，2004年。

[34] Jin-Young Kim, Seok-Jun Bu, and Sung-Bae Cho.使用基于深度自动编码器的转移生成式对抗网络进行零日恶意软件检测。*Information Sciences*, 460:83-102, 2018.

[35] Dhilung Kirat, Lakshmanan Nataraj, Giovanni Vigna, and B. S. Manjunath.Sigmal：一个基于静态信号处理的恶意软件分流。在*第29届年度计算机安全应用会议论文集中*，第89-98页。ACM，2013年。

[36] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang.在终端主机上有效和高效的恶意软件检测。在*USENIX安全研讨会上*，第351-366页，2009年。

[37] 孔德光和严冠华。基于结构信息的判别式恶意软件远程学习用于自动恶意软件分类。在*第19届ACM SIGKDD国际会议上，关于知识识别和数据挖掘的论文集*，第1357-1365页。ACM, 2013.

[38] Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna.被混淆的二进制文件的静态反汇编。在*USENIX安全研讨会上*，第13卷，第18-18页，2004。

[39] Bo Li, Kevin Roundy, Chris Gates, and Yevgeniy Vorobeychik.大规模识别恶意的单子文件。在*第七届ACM数据和应用安全与隐私会议上*，第227-238页，2017年。

[40] Scott M. Lundberg和Su-In Lee.解释模型预测的统一方法。In *Advances in Neural In-form Processing Systems*, pages 4765-4774, 2017.

[41] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini.MAMADROID：通过建立行为模型的马尔可夫链来检测安卓恶意软件。*第24届网络和分布式系统安全研讨会（NDSS）论文集*，2017。

[42] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis.无斑点沙箱：使用磨损工件规避恶意软件分析系统。*IEEE安全与隐私研讨会*，2017。

[43] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai.Kitsune：用于在线网络入侵检测的自动编码器集合。In *The Network and Distributed System Security Symposium (NDSS) 2018*, 2018.

[44] Andreas Moser, Christopher Kruegel, and Engin Kirda.恶意软件检测的静态分析的局限性。在*年度计算机安全应用会议上*，第421-430页。IEEE, 2007.

[45] James Newsome, Brad Karp, and Dawn Song.Polygraph：自动生成多态蠕虫的签名。在*IEEE安全与隐私研讨会上*，第226-241页。IEEE, 2005.

[46] 詹姆斯-纽瑟姆，布拉德-卡普，和道恩-宋。段：通过恶意训练来挫败签名学习。在*入侵检测的最新进展国际研讨会上*，第81-105页。Springer, 2006.

[47] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami.针对机器学习的实用黑盒攻击。

在*ACM亚洲计算机和通信安全会议上*，第506-519页。ACM, 2017.

[48] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 深度学习在对抗性环境中的局限性。在*IEEE欧洲安全与隐私研讨会（EuroS&P）上*，第372-387页。IEEE，2016年。

[49] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami.蒸馏作为针对深度神经网络的对抗性扰动的一种防御手段。*IEEE安全与隐私研讨会*，2016。

[50] Razvan Pascanu, Jack W. Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas.用递归网络对恶意软件进行分类。在*2015年IEEE国际声学、语音和信号处理会议（ICASSP）上*，第1916-1920页。IEEE，2015。

[51] Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Sharif.利用故意注入噪音误导蠕虫签名生成器。在*IEEE安全与隐私研讨会上*，第15-页。IEEE, 2006.

[52] Roberto Perdisci, Wenke Lee, and Nick Feamster.基于HTTP的恶意软件的行为聚类和使用恶意网络跟踪的签名生成。在*NSDI*，第391-404页，2010年。

[53] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. 通过吃整个exe来检测恶意软件。在*第三十二届AAAI人工智能会议的研讨会上*，2018。

[54] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda.检测系统仿真器。In *Information Security*, pages 1-18.Springer, 2007.

[55] Alison DeNisco Rayome.网络安全的倦怠：工作中压力最大的10个部分。*TechRepublic*，2019年5月。

[56] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi.微软恶意软件分类挑战。*CoRR*，abs/1802.10135，2018。

[57] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net：用于生物医学图像分割的卷积网络。*MICCAI*, 2015.

[58] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee.Polyunpack：自动提取解包执行的恶意软件的隐藏代码。在*年度计算机安全应用会议上*，第289-300页。IEEE, 2006.

[59] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G. Bringas.基于数据挖掘的未知恶意软件检测的操作码序列作为可执行文件的代表。*信息科学》*，231:64-82，2013.出版商：Elsevier.

[60] Igor Santos, Javier Nieves, and Pablo G. Bringas.未知恶意软件检测的半监督学习。在*分布式计算和人工智能国际研讨会上*，第415-422页。Springer, 2011.

[61] Matthew G. Schultz, Eleazar Eskin, F. Zadok, and Salvatore J. Stolfo.用于检测新的恶意可执行文件的数据挖掘方法。在*IEEE安全与隐私研讨会上*，第38-49页。IEEE, 2001.

[62] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero.AVclass：大规模恶意软件标签的工具。在*攻击、入侵和防御研究国际研讨会上*，第230-253页。Springer，2016。

[63] M.Zubair Shafiq, S. Momina Tabish, Fauzan Mirza, and Muddassar Farooq.Pe-miner：挖掘结构信息以实时检测恶意的可执行文件。在*入侵检测最新进展国际研讨会上*，第121-141页。Springer, 2009.

[64] Karen Simonyan和Andrew Zisserman.用于大规模图像识别的极深层卷积网络。 *arXiv预印本arXiv:1409.1556*，2014。

[65] 微软卫士ATP研究团队。新的机器学习模型通过筛选好的东西来挖掘出逃避性恶意软件中的坏东西。*微软安全部*，2019年8月。

[66] 微软威胁保护情报团队。微软研究人员与英特尔实验室合作，探索新的深度学习方法进行恶意软件分类。*微软安全*，2020年5月。

[67] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G. Bringas.SoK：深层包装器检查：对运行时打包器的复杂性的纵向研究。*IEEE安全与隐私研讨会*，2015。

[68] R.Vinayakumar, K. P. Soman, and Prabaharan Poornachandran.深度安卓恶意软件检测和分类。在*计算、通信和信息学进展国际会议（ICACCI）上*，第1677-1683页。IEEE, 2017.

[69] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek.反向工程师过程的观察性调查。在*USENIX安全研讨会上*，第1875-1892页，2020。

[70] David Wagner和Paolo Soto.对基于主机的入侵检测系统的模仿性攻击。在*ACM计算机和通信安全会议上*，第255-264页，2002。

[71] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 神经净化：识别和缓解神经网络中的后门攻击。In *IEEE Symposium on Security and Privacy*, 2019.

[72] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo.基于异常有效载荷的蠕虫检测和签名生成。在*入侵检测的最新进展国际研讨会上*，第227-246页。Springer, 2005.

[73] David Warde-Farley, Ian Goodfellow, T. Hazan, G. Pa-pandreou, and D. Tarlow.深度神经网络的对抗性扰动。In *Perturbations, Optimization, and Statistics*, pages 1-32.2016.

[74] Weilin Xu, David Evans, and Yanjun Qi.Feature Squeezing：检测深度神经网络中的对抗性实例。In *Network and Distributed Systems Security Symposium*, 2018.

[75] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song.基于神经网络的图嵌入的跨平台二进制代码相似性检测.*ACM计算机和通信安全会议*, 2017.

[76] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula.基于自动编码器的网络安全应用特征学习。在*国际神经网络联合会议（IJCNN）上*，第3854-3861页。IEEE，2017年。

[77] Kim Zetter. 研究人员轻松地欺骗Cylance的基于AI的杀毒软件，使其认为恶意软件是 "好东西"。https://www.vice.com/en_us/article/ 9kxp83/researchers-easily-trick-cylances- ai-based-antivirus-into-thinking-malware- is-goodware, Jul 2019.

附录A

A.1 评价1

实践评估。我们请一位有逆向工程经验的恶意软件分析师在他过去分析过的一个恶意软件（Mikey）上使用DEEPREFLECT。在我们的工具识别出的Mikey的15个功能中，该分析员发现有13个TP和2个FP。他注意到，DEEPREFLECT发现了一个有趣的组件，他错过了这个组件，而且这两个FP被放在组件的优先级排名的底部。

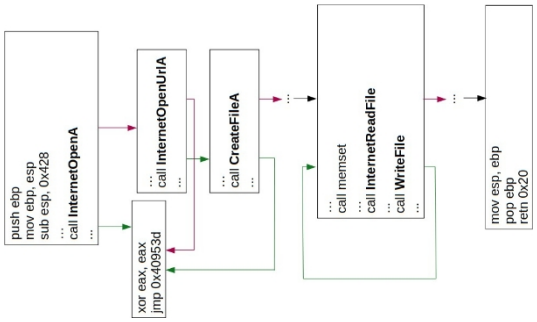

图6：指挥和控制：入侵工具传输。恶意软件通过InternetOpenUrlA()访问一个URL，通过CreateFileA()创建一个文件，并通过InternetReadFile()和WriteFile()将从连接处收到的数据写入该文件中。

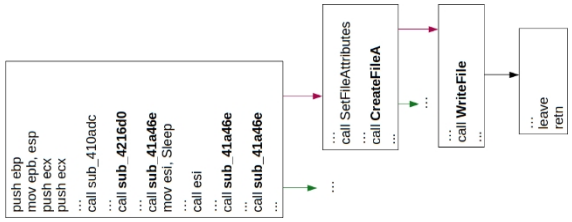

图7：防御规避：解除混淆/解码文件或在形成。这个函数对内部函数（加粗）进行了许多调用，其中包含对数据进行复杂的位操作（类似于图9）。这些复杂的操作表现出脱混淆的行为。在调用这些函数后，它通过CreateFileA()和WriteFile()将解码后的数据写入一个文件中。

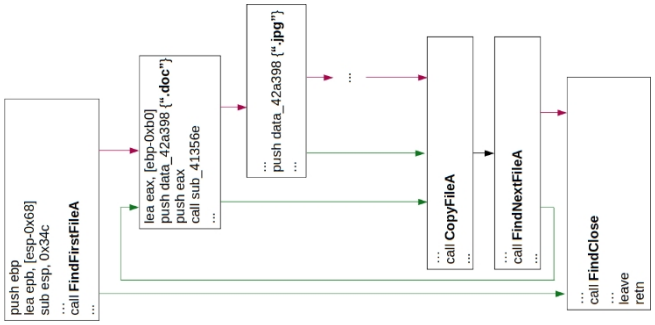

图8：发现：文件和目录发现。该功能搜索具有特定扩展名的各种文件（即doc、jpg等）。然后它将这些文件复制到一个单独的位置。这种行为可能是为其他恶意行为（如数据渗出或赎金）进行的设置。

A.2 评价2

我们发现的大多数良性功能是内存分配、加载库、从进程文件的资源部分加载数据、终止一个进程（没有上下文）等。分析师将什么标记为恶意行为可能是主观的，并依赖于他们的经验和能力，将其与MITRE ATT&CK中的描述相匹配。

发现问题	59 防卫规避	17特权升级	4	执行	11 指挥和控制7
系统信息	发现16 解密/解码文件或信息11 创建或修改系统进程2 计划任务/工作				7 应用层协议4
文件和目录发现	12 修改注册表	4 访问令牌的操纵	1 命令和脚本 Int	解释器2入口工具转移	3
应用窗口发现	9 隐藏文物	1 工艺注射	1 系统服务	2	
查询登记处	7 虚拟化/沙盒规避	1			
虚拟化/沙盒规避	5				
过程发现	4				
系统时间发现	3				
域名信任发现	1				
软件发现	1				
系统网络连接发现1					

持久性	2 影响	2 渗出	1 采集	2
外部远程服务	1 数据操纵	1 自动渗出	1屏幕捕捉	1
不详	1 网络拒绝服务	1		

表3：MITRE ATT&CK类别和子类别的计数，由分析师在§4.4中发现。

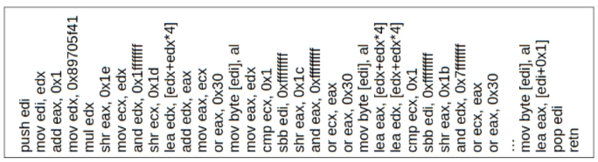

图9：防御规避：解密/解码文件或信息。这个函数对数据进行各种位运算。像这样复杂的逻辑可以被理解为执行一些去混淆或解码，以努力隐藏恶意软件 解释或收集的数据。

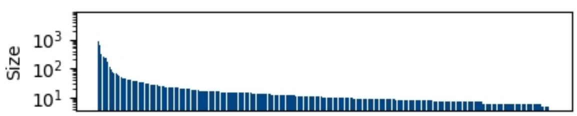

图10：使用DEEPREFLECT在我们的恶意软件数据集上的集群大小分布。X轴是每个集群的ID。

A.3 评价4

新颖的恶意软件家族。我们选择了四个著名的恶意软件家族：Zbot、Gandcrypt、cosmicduke和wannacry。对于Zbot，之前有22,433个集群，之后有22,470个集群。样本存在于359个集群中，其中4个只有zbot，其他355个是混合的。平均而言，统一集群中的zbot样本数量为5.75，混合集群中的数量为1.49。也就是说，有4个新概念原本不在旧的聚类中。320个新聚类（包含zbot）与旧聚类相同。也就是说，320个聚类（如果有标签）会自动提供320 x 1.49 = 476.8个功能标签，让分析员去审查较新的聚类（行为）。有18个新聚类只包含旧的噪声点的样本的情况。有187个新聚类包含旧的噪声点。最后，有17个新集群包含了zobt样本，被分成了两个集群（即与旧集群不一样）。类似的观察也发生在其他族。值得注意的是，cosmicduke样本并没有产生新的概念（即只由该族组成的新聚类），而加入wannacry后的大部分新聚类是，这些样本是旧的噪声点组成。

A.4 评价5

混淆。首先，我们通过DEEPREFLECT运行了所有五个（加上没有启用混淆功能的*ollvm*编译的原始源代码），观察它使用为聚类选择的阈值识别的函数。我们原始的、未被混淆的样本有158个函数被突出显示，A有118个，B有156个，C有138个，D有118个，E有137个。我们没有对825个函数进行人工检查，而是从每个样本中随机选择了10%的函数进行标注（我们选择10%是因为它可以确保我们有足够的统计学意义来依赖我们的结果--我们确定了42%的良性函数和57%的恶意函数，误差率为11%）。我们未被混淆的样本有12个良性功能和4个恶意功能被强调。我们的真实标签比我们对评估集的标签更严格，12个良性功能中的7个可能是由MITRE标记的。A有5个良性的和7个恶意的功能。然而，5个良性功能中的2个可以由MITRE描述。B有10个良性的和6个恶意的功能。然而，在10个良性功能中，有3个可以被MITRE描述。C有4个良性功能和10个恶意功能，但是4个良性功能中的2个可以被MITRE描述。D有2个良性的和10个恶意的功能。没有一个良性功能可以被MITRE描述。最后，E有3个良性的和11个恶意的功能。然而，3个良性功能中的一个可以由MITRE描述。最后，我们对突出的功能进行聚

类，以观察它们对其他功能的影响。我们假设了两种结果：（1）被混淆的函数看起来非常模糊，以至于它们被标记为噪声点，或者（2）被混淆的函数看起来一致地模糊，所以它们被集中在一个大的集群下。然而，我们没有看到这两种情况。