DeepReflect：通过二进制重构发现恶意行为（恶意代码ROI分析经典）

> 该论文收录于USENIXSec21

原文链接：https://www.usenix.org/conference/usenixsecurity21/presentation/downing

- 姓名：管昀玫
- 学号：2013750
- 专业：计算机科学与技术

一、摘要

深度学习已在恶意软件分类任务中表现出良好的结果。然而：

- **人工分析效率低**：对于未知恶意软件的binary，分析人员仍要花大量时间来利用静态分析工具逆向整个binary，从而识别关键的恶意行为。
- **监督学习开销大**：尽管机器学习可用来帮助识别二进制的重要部分，但由于获取足够大的标记数据集开销很大，因此监督学习方法是不切实际的。

为了提高静态（或手动）逆向工程的生产力，作者提出了DeepReflect：一种用于定位和识别恶意二进制文件中恶意软件组件的工具。

1. DeepReflect采用了一种新颖的方法来定位恶意软件中的恶意组件（函数），首先使用无监督的深度神经网络来确定它们的位置。
2. 然后，通过半监督聚类分析对这些恶意组件进行分类，分析人员在日常工作流程中逐步提供标签，根据恶意行为对恶意函数进行分类。
3. 这个工具非常实用，因为它不需要有标记的数据来训练定位模型，也不需要最小的或非侵入性的标记来增量地训练分类器。

作者对超过26,000个恶意软件样本进行了评估，邀请了5名恶意软件分析人员参与。实验结果表明，DeepReflect平均减少了每位分析人员需要逆向工程的函数数量的85%。与基于签名的工具CAPA相比，本方法还能检测到80%的恶意软件组件，而CAPA仅能检测到43%。

此外，DeepReflect提出的自动编码器比Shap（一种人工智能解释工具）表现更好。这一点非常重要，因为Shap是一种先进的方法，需要一个有标记的数据集，而作者的自动编码器则不需要。

通过DeepReflect工具，分析人员可以提高静态逆向工程的生产力，减少手动分析的工作量，并有效地定位和识别恶意软件二进制文件中的恶意组件。

二、引言

2.1 背景引出挑战

静态逆向工程恶意软件是一个手动且乏味的过程。公司每周可以收到多达 500 万个PE样本。 虽然大多数组织提前对这些样本进行分类（triage），以减少要分析的恶意软件数量（即，检查 VirusTotal来获取反病毒 (AV) 引擎结果、在受控沙箱中执行样本、提取静态和动态签名等），但最终仍然需要静态逆向工程的恶意软件样本。这是因为总会有新的恶意软件样本，它们尚未被反病毒公司分析过，或者缺乏用于识别这些新样本的签名。而且，这些样本最终可能会被拒绝在分析人员的动态沙箱中执行。

目前的解决方案主要是通过创建签名、分类和聚类恶意软件样本来进行恶意软件分析。然而，这些解决方案只能预测样本的类别（例如，良性或恶意，或者特定的恶意软件家族），**无法定位或解释恶意软件样本内部的行为（例如，定位恶意函数的位置、解释恶意函数的行为）**。而恶意软件分析师需要执行这些行为来生成报告并改进其公司的恶意软件检测产品，且工作量巨大。

为了了解他们的需求，作者咨询了四名逆向工程恶意软件的分析师（一名来自反病毒公司，三名来自政府部门）。研究发现，如果恶意软件分析师拥有一个工具，可以：

- **确定恶意软件中恶意函数的位置**
- **标记这些恶意函数的行为**

那么，他们的工作将更加高效。然而，开发这样一种工具面临以下挑战：

- 需要能够区分良性和恶意行为
- 理解识别出的恶意行为的语义

对于第一个挑战，区分良性和恶意行为是困难的，因为恶意软件和良性软件的行为通常在高层次上重叠。对于第二个挑战，自动标记和验证这些行为是困难的，因为没有针对单独标记恶意软件函数的数据集（与使用反病毒标签的公开数据集中的恶意软件检测和分类系统不同）。

2.2 如何解决挑战

为了解决这些挑战，作者开发了DEEPREFLECT，它使用：

- 一个无监督的深度学习模型来定位二进制中的恶意函数
- 一个半监督聚类模型，它使用从分析人员的日常工作流程中获得的少量标签对识别的函数进行分类

为了定位（locate）二进制文件中的恶意软件组件，作者使用自动编码器(autoencoder，AE)。AE是一种基于神经网络的机器学习模型，其任务是将其输入重构为输出（编码还原）。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。作者的直觉是，如果在良性二进制文件上训练AE，它将很难重建恶意二进制文件（即作者没有训练它的样本）。自然来说，AE将无法重建（reconstruct）包含恶意行为的二进制数据区域（在良性样本中是不可见或罕见的）。因此，通过检测重构错误，可以识别恶意软件中的恶意组件。另外，由于自动编码器是无监督训练的，作者不需要大量标记的样本，而可以利用自身的恶意软件二进制数据集。

为了对定位的恶意软件组件进行分类，作者：

- 对恶意软件样本中所有已识别的函数进行聚类
- 使用分析人员在日常工作流程中所做的注释（即少量人工分析的函数行为标签）来标记聚类结果

这种方法是半监督的，因为每个类簇只需要少数函数的行为标签（如三个）即可将大多数标签分配给整个集群。随着时间推移，作者可以将AE识别的函数映射到聚类模型来预测函数的类别（如，C&C、特权升级等），即认为函数和最接近的类簇有相同的行为标签。这反过来又节省了分析人员的时间，因为他们不必一次又一次地对相同的代码进行逆向工程。

注意，无监督 AE 为恶意软件分析人员提供了即时实用程序，无需训练或使用半监督聚类模型。这是因为它：

- 通过对最相关的函数进行排序（重构误差）来吸引分析师的注意力
- 过滤掉可能需要花费分析师数小时或数天时间来解释的函数

DEEPREFLECT根据作者是为恶意软件分析人员的反馈进行设计和修改的，并评估其有效性和实用性。

作者评估了DEEPREFLECT的性能，包括五个工作：

1. 识别恶意软件中的恶意活动
2. 聚类相关的恶意软件组件
3. 将分析人员的注意力集中在重要事情上
4. 揭示不同恶意软件家族之间的共享行为
5. 处理涉及混淆的对抗性攻击

2.3 作者的贡献

作者的贡献如下：

- 提出了一个新颖的工具，它可以帮助恶意软件分析师：
    1. 在静态恶意软件样本中自动定位和识别恶意行为。
    2. 洞察分析不同恶意软件家族之间的功能关系。
- 提出一种在静态分析中使用机器学习的新颖实用方法：
    1. AE训练是在一种无监督方式下进行的，无需为系统标注任何样本，就可以产生突出显示恶意软件组件的实用程序。
    2. 分类是以半监督方式完成，具有最小的干预：分析人员的常规工作流的注释用作标签，群集中的大多数标签用于对相关的恶意软件组件进行分类。
- 本文提出了一种解释框架定位恶意软件重要部分的方法，该方法可以映射回原始二进制或控制流图的特征。

三、Scope & Overview

3.1 Motivation

图1展示了一个典型的恶意软件分析师Molly的工作流程。当给定一个恶意软件样本，Molly的任务是了解该样本在做什么，以便她写一份技术报告并改进公司的检测系统，从而在未来识别该类样本。

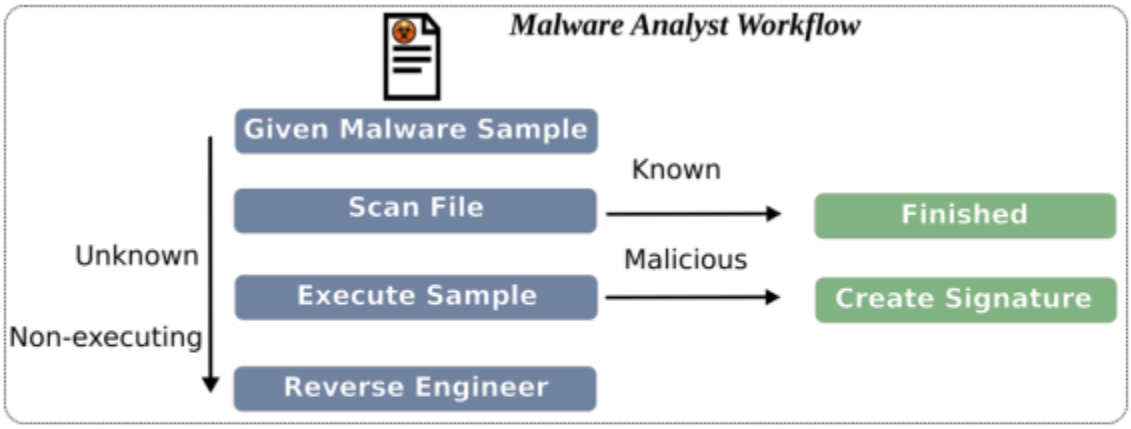

Figure 1: The general workflow of a malware analyst. DEEPREFLECT assists the analyst when they must statically reverse engineer an unknown malware sample.

1. 首先查询VT（VirusTotal）和其他组织，以确定他们以前是否见过这个特定的样本；如果是一个已被恶意样本库收录的应用，则结束本次工作。
2. 在一个自定义的沙箱中执行样本以了解其动态行为，然而没有显示任何恶意行为或拒绝执行；运行一些内部工具，诱使恶意软件执行其隐藏的行为，但仍无效时。
3. 尝试脱壳（unpacking）和静态逆向分析恶意样本，以了解其潜在行为。
4. 在反汇编程序（IDA Pro 或 BinaryNinja）中打开脱壳后的样本，被数千个函数淹没，接着运行各种静态签名检测工具来识别恶意软件的某些特定恶意组件，但仍无效。
5. 逐个查看每个函数（可能通过 API 调用和字符串过滤）以尝试了解它们的行为。
6. 在分析样本的行为后，撰写分析报告（包含基本信息、失陷标示IOCs、静态签名等）。

设计目标：

对于每一个恶意样本，Molly 都需要付出重复的劳力成本，使得这项工作冗余又耗时。本文的 DeepReflect 设计了两种功能减轻恶意应用分析师的工作：（1）定位：缩小需要检查的函数的范围，从上千个到包含了最有可能是恶意代码的子集；（2）标注：找出已知的较为相似的函数，给出标签。

> Indicators of compromise (IOCs) *are "pieces of forensic data, such as data foumorend in system log entries or files, that identify potentially malicious activity on a system or network."*

3.2 Proposed Solution

作者提出了DEEPREFLECT，该工具能：

1. 定位恶意软件binary中的恶意函数
2. 描述这些函数的行为

虽然分析人员可能首先尝试通过搜索特定的字符串和API调用来静态地识别行为，但这些行为很容易被分析人员混淆或隐藏。DEEPREFLECT没有做出这样的假设，并试图通过**CFG特性和API调用**的组合来识别这些相同的行为。

DEEPREFLECT通过学习正常情况下良性的二进制函数来工作。因此，任何异常都表明这些函数不会出现在良性二进制文件中，而可能被用于恶意行为中。这些异常函数更可能是恶意函数，分析师可以只分析它们，从而缩小工作范围。如图5所示，DEEPREFLECT将分析师必须分析的函数数量平均减少了 85%。此外，实验表明作者的方法优于旨在实现相同目标的基于签名的技术。

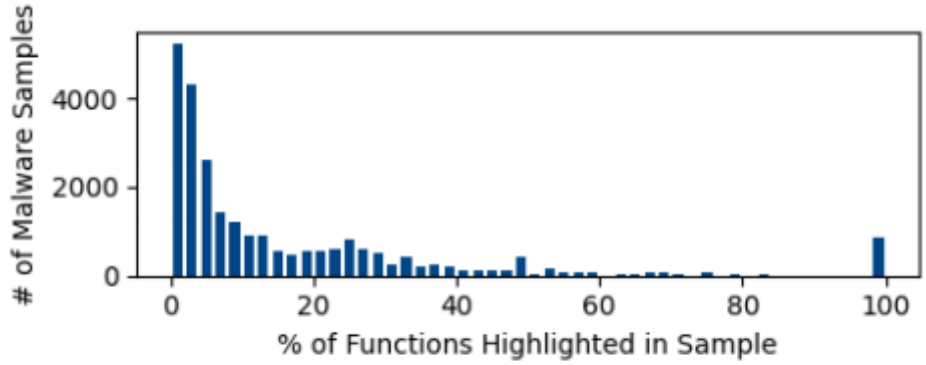

Figure 5: Function Counts. Percentage of functions (per malware sample) the analyst has to review.

3.3 Research Goals

分析人员需要定位和描述恶意软件样本内部功能的行为。DEEPREFLECT有四个主要目标：

- G1：准确地识别恶意软件样本中的恶意活动
- G2：帮助分析人员在静态分析恶意软件样本时集中注意力
- G3：处理新的（不可见的）恶意软件家族
- G4：深入了解恶意软件家族的关系和趋势

四、模型设计

4.1 总体框架

DEEPREFLECT的目标是识别恶意软件二进制中的恶意函数。在实践中，它通过定位异常基本块来识别可能是恶意的函数。然后，分析人员必须确定这些函数是恶意行为还是良性行为。DEEPREFLECT有两个主要步骤，如图2所示：

- RoI检测（RoI detection）：将二进制文件表示为特征矩阵，对应每个basic block（BB）的特征向量，使用 AutoEncoder 在良性应用上做无监督学习；对于任意输入样本，重构误差较大的特征向量对应的基本块（RoI）可能包含恶意代码；
- RoI注释（RoI annotation）：将函数表示为其包含的 ROI（regions of interest） 特征向量之和，使用 HDBSCAN 对恶意应用中 RoI 不为空的函数做聚类；对于输入函数，选择最相似（中心距离最小）的簇中的绝大多数标签作为函数标签。

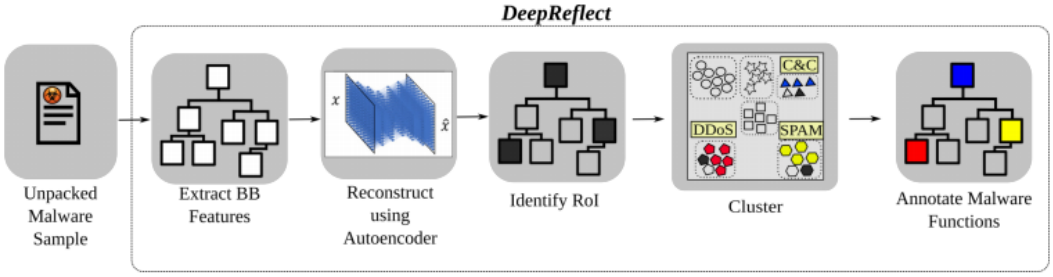

Figure 2: Overview of DEEPREFLECT. Our system takes unpacked malware samples as an input, extracts CFG features from each input (basic block (BB)), applies them to a pretrained autoencoder model to highlight RoI (*regions of interest*). Finally, it clusters and labels these regions.

(1) 术语 Terminology

首先定义恶意行为（malicious behaviors）的含义。作者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ground-truth。通过MITRE ATT&CK框架描述，如表3所示。

Discovery	59	Defense Evasion	17	Privilege Escalation	4	Execution	11	Command and Control	7
System Information Discovery	16	Deobfuscation/Decode Files or Information	11	Create or Modify System Process	2	Scheduled Task/Job	7	Application Layer Protocol	4
File and Directory Discovery	12	Modify Registry	4	Access Token Manipulation	1	Command and Scripting Interpreter	2	Ingress Tool Transfer	3
Application Window Discovery	9	Hide Artifacts	1	Process Injection	1	System Services	2		
Query Registry	7	Virtualization/Sandbox Evasion	1						
Virtualization/Sandbox Evasion	5								
Process Discovery	4								
System Time Discovery	3								
Domain Trust Discovery	1								
Software Discovery	1								
System Network Connection Discovery	1								
Persistence	**2**	**Impact**	**2**	**Exfiltration**	**1**	**Collection**	**2**		
External Remote Services	1	Data Manipulation	1	Automated Exfiltration	1	Screen Capture	1		
Unknown	1	Network Denial of Service	1						

Table 3: The counts of MITRE ATT&CK categories and subcategories found by the analysts in §4.4.

然而，当静态逆向工程评估恶意软件二进制文件时（即在野恶意软件二进制 in-the-wild malware binaries），作者有时无法肯定地将观察到的低级函数归因于更高级别的描述。例如，恶意软件可能会因为许多不同的原因修改注册表项，但有时确定哪个注册表项因什么原因而被修改是很困难的，因此只能粗略地标记为`防御逃避：修改注册表（Defense Evasion: Modify Registry）`。即使是像CAPA这样的现代工具，也能识别出这些类型的模糊标签。因此，在作者的评估中，作者将"恶意行为"表示为可由MITRE ATT&CK框架描述的函数。

(2) RoI Detection

检测的目标是自动识别恶意软件二进制文件中的恶意区域。例如，作者希望检测C&C逻辑的位置，而不是检测该逻辑的特定组件（例如，网络API调用connect()、send() 和 recv()）。RoI检测的优点是分析人员可以快速定位启动和操作恶意行为的特定代码区域。先前的工作只关注于创建临时签名，简单地将二进制文件标识为恶意软件或仅基于API调用的某些函数。这对于分析人员扩大他们的工作特别有用（即不仅仅依赖手动逆向工程和领域专业知识）。

(3) RoI Annotation

注释的目标是自动标记包含RoI的函数的行为，即识别恶意函数在做什么。由于分析人员为标记集群所执行的初始工作是一个长尾分布。也就是说，只需要前期做比较重要的工作，随着时间推移，工作量会减少。这个过程的优点很简单：它为分析人员提供了一种自动生成未知样本的报告及见解的方法。例如，如果恶意软件示例的变体包含与之前的恶意软件示例相似的逻辑（但对于分析人员来说看起来不同以至于不熟悉），作者的工具为他们提供了一种更快实现这一点的方法。

4.2 RoI Detection

训练在良性分布上的自编码器，可利用重构误差来识别恶意行为（异常）。作者的假设是，与良性二进制文件相比，恶意软件二进制文件将包含相似但独特的功能。

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

得到一个标量，再与给定的阈值做比较。

但考虑到 malware 与 goodware 有相似但独特的功能，因此选择识别恶意的**区域**而非整个样本。MSE 由局部均方误差（LMSE）代替。与先前识别整个样本为恶意区域的工作相比，作者识别了每个样本中的恶意区域。作者计算的 `localized MSE` 定义如下：

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

得到一个向量，再将其中的平方误差逐个与阈值进行比较，大于阈值的块被识别为异常。

我们把在样本x中识别的ROI的映射集合表示为：

$$R_x = \left\{ x^{(i)} \middle| \left(x^{(i)} - \hat{x}^{(i)}\right)^2 \geq \phi \right\} \quad (3)$$

(1) Features

作者特征（c）的灵感来自于先前工作中发现的特征，即属性控制流图（attributed control flow graph，ACFG）特征。在这些工作中，ACFG特征被选择来执行二进制相似性，因为它们假设这些特征(由结构和数字CFG特征组成)将在多个平台和编译器上是一致的。

- Genius
- Gemini

为了在二进制样本中定位恶意行为的位置，编码使用的特征必须一对一的映射回原样本。区域由 BB 界定，即 m 表示 BB 的个数；每个BB的特征用 c 个特征值表示，因此输入样本 x 被表示为一个 $m \times c$ 的矩阵，该矩阵使用c个静态特征捕获前m个基本块以总结样本的行为。m设置为20k个基本块，是因为95%的数据集样本具有20k或者更少的基本块， c设置为18个特征。

- 结构（*2）：后继节点个数，介数中心性

  > 对于连通图中的每一对节点，在节点之间至少存在一条最短路径，使得路径通过的边数(对于未加权图)或者边权重的和(对于加权图)最小。节点的中心性是**经过该节点的最短路径的数量**。

  结构特征2个，每个基本块的后代数量和betweenness score，可以描述不同功能的控制流结构，比如网络通信（connect, send, recv）或文件加密（findfile, open, read, encrypt, write, close）。如图6所示。

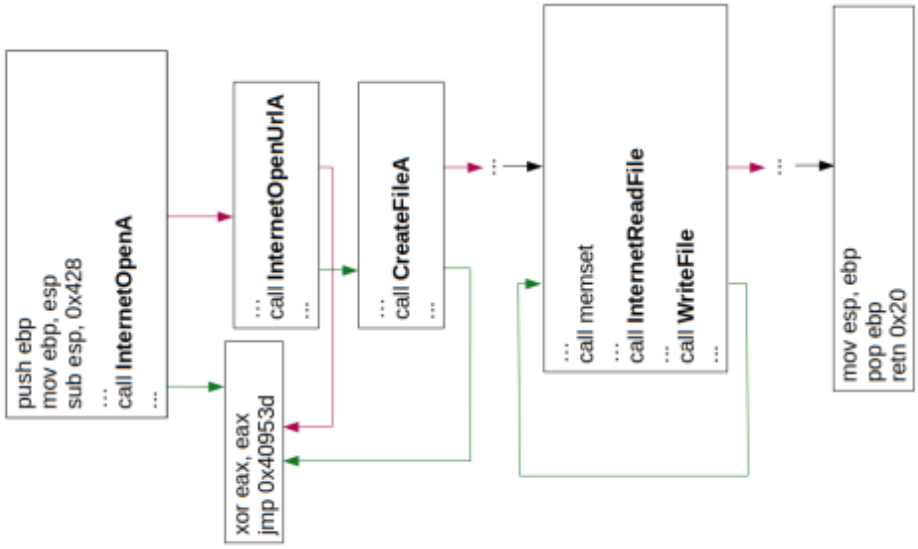

Figure 6: Command and Control: Ingress Tool Transfer. The malware accesses a URL via `InternetOpenUrlA()`, creates a file via `CreateFileA()` and writes data received from the connection to the file via `InternetReadFile()` and `WriteFile()`.

- 算数指令

  算术指令3个，每个基本块基本数学、逻辑运算、位移指令的数量（“basic math”, “logic operation”, and “bit shifting”）。这些算术指令特征可以用来表示如何对更高层次的行为执行数学运算，以及数字如何与函数交互。例如，加密函数可能包含大量的XOR指令，混淆函数可能包含逻辑和位移操作的组合等。

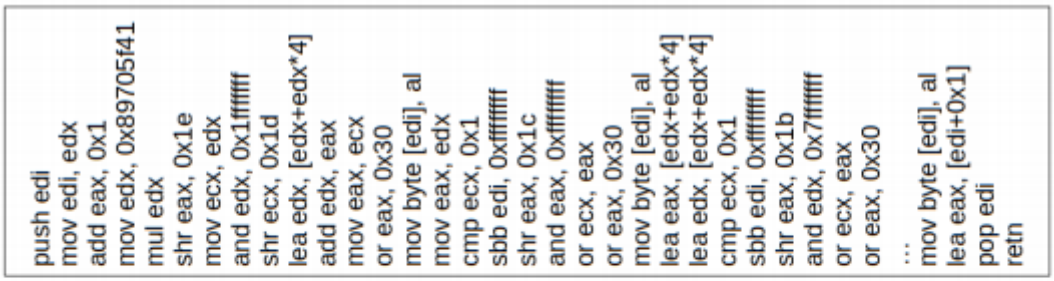

Figure 9: Defense Evasion: Deobfuscate/Decode Files or Information. This function performs various bitwise operations on data. Complex logic like this could be construed as performing some deobfuscation or decoding in an effort to hide data the malware interprets or gathers.

- 转移指令

  转移指令3个，每个基本块内堆栈操作，寄存器操作和端口操作的数量（“stack operation”, “register operation”, and “port operation”）。这些底层特征可描述更高级别函数的传输操作，比如函数的参数和返回值是如何与函数内其余数据交互的，从而描述更复杂的逻辑和数据操作。例如去混淆、解密函数可能设计move-related指令，C&C逻辑设计更多堆栈相关指令。

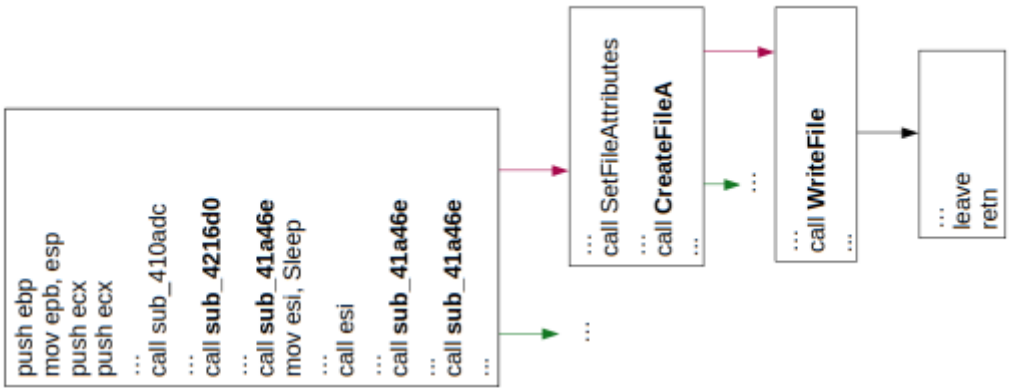

Figure 7: Defense Evasion: Deobfuscate/Decode Files or Information. This function makes many calls to internal functions (bolded) which contain complex bitwise operations on data (similar to that of Figure 9). These complex operations exhibit deobfuscation behavior. After calling these functions, it writes the decoded data to a file via `CreateFileA()` and `WriteFile()`.

- API Call Categories

  API类别10个， 包括"filesystem", “registry”, “network”, “DLL”,“object”, “process”, “service”, “synchronization”, “system information”, and "time"相关的API调用数量。调用不同类型API可执行不同类型功能，直接的表示了高层的函数行为，是很关键的特征。

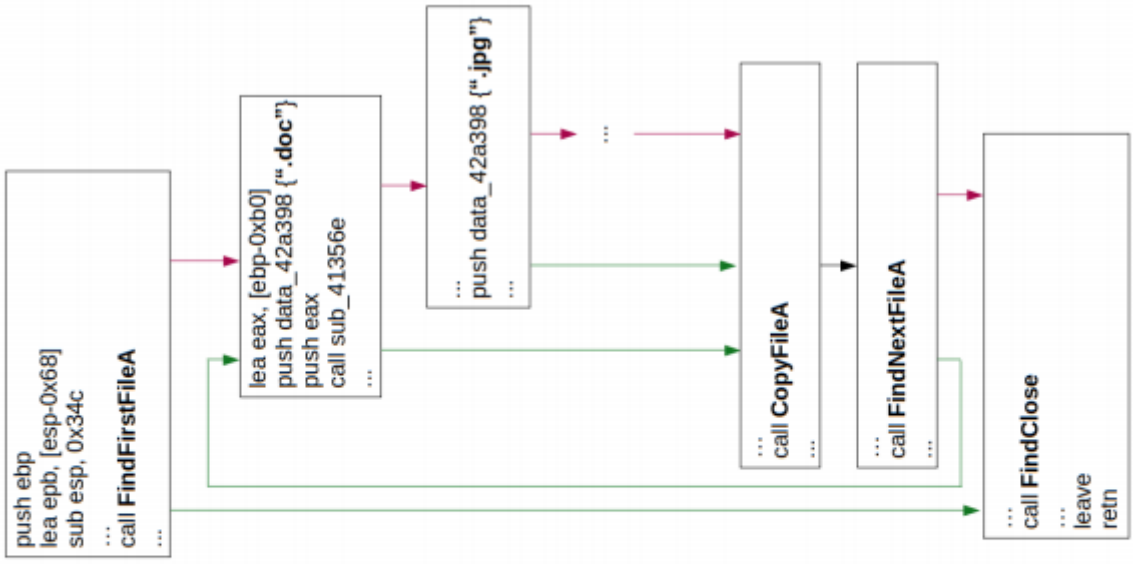

Figure 8: Discovery: File and Directory Discovery. This function searches for various files with specific extensions (i.e., doc, jpg, etc.). It then copies those files to a separate location. This behavior could be a setup for additional malicious behaviors like data exfiltration or ransom.

> 本文工作API特征的选择受到先前恶意软件检测工作[18]的启发。本文使用的ACFG特征比Genius和Gemini更细致。本文没有用字符串特征，因为容易被混淆、隐藏。

inspired from **ACFG features**:

种类	特征名
统计特征	字符串常量
	数值常量
	传送指令数目
	调用指令数目
	指令数目
	算数指令数目
结构特征	子节点的数目
	介数中心性

区别：

1. 指令的类别进一步细化（finer-grained）
2. 新增了API类别相关的特征（summarize program behavior）
3. 舍去了与 字符串/数值常量 相关的特征（too easily prone to evasion attacks）。

(2) Model

AutoEncoder使用U-Net模型，U-Net的优点是其在编码器和解码器之间有跳过连接，对样本`x`可以跳过某些特征的压缩以在重构的`x'`中保持更高的保真度。

首先收集大量的良性样本，对每个binary抽取上述18个静态特征用于表示该binary。设有用feature表示的样本x，AE重构后得到x'，训练的目标是最小化重构损失，即输入x和输出x'之间的损失。

$$\mathcal{L}_2(x,\hat{x}) = \sum(x-\hat{x})^2 \qquad (4)$$

RoI Detection会在m个基本块中检测出一些异常基本块。这些基本块分别属于不同的函数，使用例如BinaryNinja的工具就可以确定ROI属于哪些函数，即认为这些函数可能是恶意函数，也就完成了恶意函数定位的任务。后续RoI Annotation就是对这些函数聚类，完成恶意函数行为标记（分类）的任务。

4.3 RoI Annotation

给定一个新样本`x`，作者希望识别其每个函数的行为（类别），并将其报告给Molly。由于标记所有的函数都是不实用的，所以作者只注释了少量的函数，并使用聚类分析来传播结果。

(1) Clustering Features

假设一组脱壳恶意软件，按上述特征提取方式（18种特征）得到每个binary的特征表示，其中一个binary为x。

F是BinaryNinja找到的x中的函数集合。对于每个$f_i \in F$，q_i是f_i中的ROI，$q_i \subset R_x$.

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$qi_i \neq \emptyset$，我们将f_i的行为总结为$\frac{1}{q_i}\sum q_i$，并把它加入D。重复这一操作，直到所有的恶意软件都在我们的集合中。

(2) Clustering Model

使用PCA将特征数从18降维至5，然后使用HDBSCAN算法对5维特征聚类。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，即DBSCAN）

4.4 Deployment

接下来，作者将描述如何部署和使用它。

(1) 初始化 Initialization

1. 首先对良性和恶意binaries脱壳
2. 提取binary静态特征，形成20×18的矩阵
3. 用良性样本训练AutoEncoder
4. 使用训练好的AE从恶意样本中提取ROIs，即恶意基本块位置
5. 计算恶意二进制中恶意函数的行为表示，加入聚类的训练集D
6. PCA降维并聚类生成C

人工分析恶意软件手动打标，这些label注释到聚类训练集中，从而评估实验结果。换句话说，每个cluster只需要其中几个函数的label，就可确定整个cluster的label，即确定整个cluster中函数的恶意行为。

(2) 执行 Execution

当Molly收到一个新的样本`x`，DeepReflect会自动定位恶意函数并标注恶意行为。

- 对样本x执行脱壳（unpack）
- 通过AutoEncoder获取ROIs
- 使用BinaryNinja以及ROIs确定恶意函数集合，然后计算恶意函数的行为表示
- PCA模型降维
- 计算每个恶意函数最相近的集群，通过计算和聚类中心的距离实现
- 分配大数据集群注释给函数

接下来，Molly分析highlighted functions，从而实现：

- obtains a better perspective on what the malware is doing
- annotates any function labeled “unknown” with the corresponding MITRE category (dynamically updating D)
- observe shared relationships between other malware samples and families by their shared clusters（共享关系，分析恶意软件家族的相关性）

五、实验评估

五个方面：

- Reliability: 在三个恶意样本（rbot, pegasus, carbanark）上标注函数（malicious/benign）作为 ground truth，与三个 baseline 比较 AUC
- Cohesiveness: 随机抽取样本并让恶意应用分析师标注，与 DeepReflect 的结果作比较
- Focus: DeepReflect 返回的结果，使得恶意应用分析师需分析函数数量的减少
- Insight: 不同恶意应用家族之间共享的功能，以及 DeepReflect 处理未知恶意应用家族的能力
- Robustness: 通过混淆、修改恶意应用源码测试逃逸 DeepReflect 的成功率

5.1 Dataset

前提：能使用 Unipacker 成功脱壳，根据文件内容的哈希去重

良性样本：从CENT爬取，包含了 22 个类别，共 23,307 个

恶意样本：VirusTotal上2018年收集的，包含 4,407 个恶意应用家族，36,396 个PE文件

Category	Size	Category	Size
Drivers	6,123	Business Software	1,692
Games	1,567	Utilities	1,453
Education	1,244	Developer Tools	1,208
Audio	1,023	Security	1,000
Communications	994	Design	844
Digital Photo	826	Video	787
Customization	778	Productivity	730
Desktop Enhancements	699	Internet	695
Networking	612	Browsers	440
Home	390	Entertainment	257
Itunes	43	Travel	17

Table 1: Benign Dataset: 22 categories from CNET.

Label	virut	vobfus	hematite	sality	crytex
Size	3,438	3,272	2,349	1,313	914
Label	wapomi	hworld	pykspa	allaple	startsurf
Size	880	720	675	470	446

Table 2: Malware Dataset: Top 10 most populous families.

特征18个：Inspired from ACFG features used for bug-finding (CCS 2017)

- Structural: Flow of operations (e.g., connect, send, recv, etc.)
- Arithmetic Instruction Types: How mathematical operations are carried out at the higher level(e.g., encryption, obfuscation)
- Transfer Instruction Types: Flow of data (arguments provided to and returned from functions)
- API Call Categories: Used to execute behaviors (filesystem, registry, network, process, etc.)

5.2 Evaluation 1 – Reliability（可靠性）

为了评估DeepReflect自动编码器的定位能力，作者使用的baseline如下：

- 利用 VGG19 做监督学习，为了模型能收敛，使用去了常量的 ACFG 特征（记作 ABB 特征），预测样本的恶意应用家族，使用 SHAP 可解释机制输出基本块特征重要性，函数重要性由基本块 SHAP 值加和（负值置零）；

  > **SHAP** (**SH**apley **A**dditive ex**P**lanation): 观察某一个（组）样本的预测中各个特征对预测结果产生的影响沙普利法定义i加入组织S的**边际贡献**（marginal contribution）：
  > $$\delta i(S) = v(S \cup \{i\}) - v(S)$$
  >
  > Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems, pages 4765–4774, 2017.

- CAPA: FireEye 开发的基于签名的、从二进制文件中识别恶意行为的工具；

> (FPR, TPR)是确定的，ROC 曲线是三点连线
>
> https://github.com/fireeye/capa

- FunctionSimSearch: Google Project Zero 开发的函数相似性检测工具，比较方法是使用训练数据集中良性样本训练模型，再用三个样本中有 ground truth 的函数查询，取前 1,000 个最相似的函数的相似度绘制 ROC 曲线。

> 理想状态：恶意函数的相似度远小于良性函数
>
> https://github.com/googleprojectzero/functionsimsearch.

静态的分析了三个恶意软件的源代码（rbot, pegasus, carbanak），分析了其中恶意组件的位置。结果如Figure 3，横线为80% True Positive Rate。

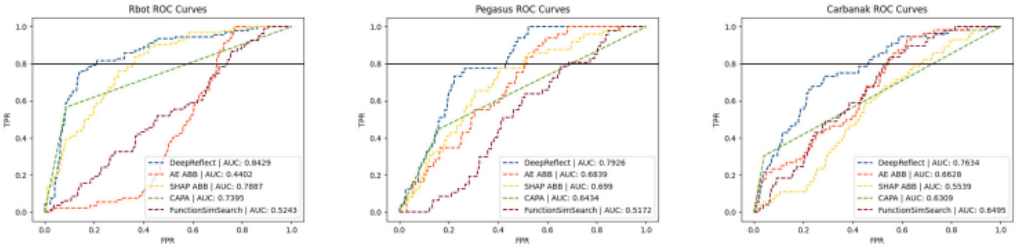

Figure 3: The ROC plot (performance at every threshold) for DEEPREFLECT, AE using ABB features, SHAP using ABB features, CAPA, and FunctionSimSearch on the three ground truth malware samples. The horizontal black bar represents a TPR of 80%.

5.3 Evaluation 2 – Cohesiveness（凝聚）

FPR 选定5%（在ground truth数据集上表现为“TPR/FPR of 40%/5%”）来确定阈值，自编码器 M 在 25,206 个恶意样本（< ~36k）中识别出了 593,181 恶意函数。聚类得到了 22,469 个簇，最大的包含了 6,321 个函数，而最小的包含5个，此外还有 59,340 个噪点。

在图10中，作者展示了类簇大小上的分布。图中显示，存在一个长尾分布（这在基于密度的聚类中很常见），其中最多的前10个集群占函数的5%。

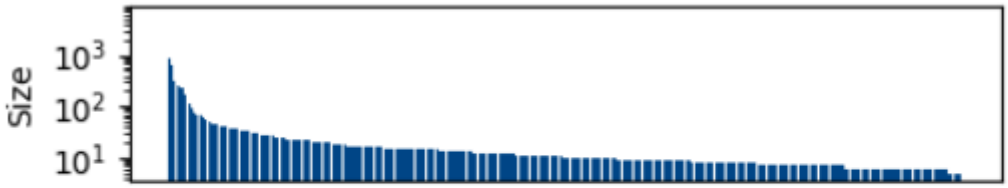

Figure 10: Cluster size distribution on our malware dataset using DEEPREFLECT. The x-axis is each cluster ID.

逆向工程师：5 个，有 2-7 年逆向经验。从 25 个有最大 RoI 的恶意应用家族中随机抽取了 177 个函数，使用 MITRE ATT&CK 定义的类别进行标注，手动聚类了 78 个簇，其中每个函数的分析时间在 15-30 分钟。

比较结果中，有 5 个人工分类的簇在 C 中属于不同的簇，有 8 个不同的在 C 中被错误的合并；有 89.7% 的函数结果是匹配的。

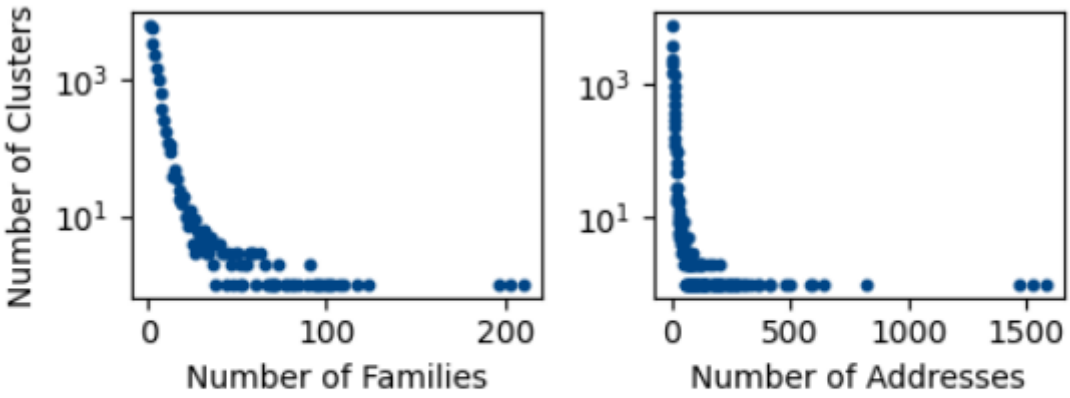

Figure 4: Cluster Diversity. Left: the distribution of families per cluster. Right: the distribution of addresses per cluster to show that there is no bias in function location.

此外，聚类质量存在问题，相同功能却被聚集在不同类簇中，分析了3个案例，主要因为小地方存在差异，聚类算法过于敏感。

5.4 Evaluation 3 – Focus

DeepReflect缩小需要人工分析的函数的范围的能力。如图5所示，很多样本需要分析的函数数量降低了90%以上。平均降低85%。

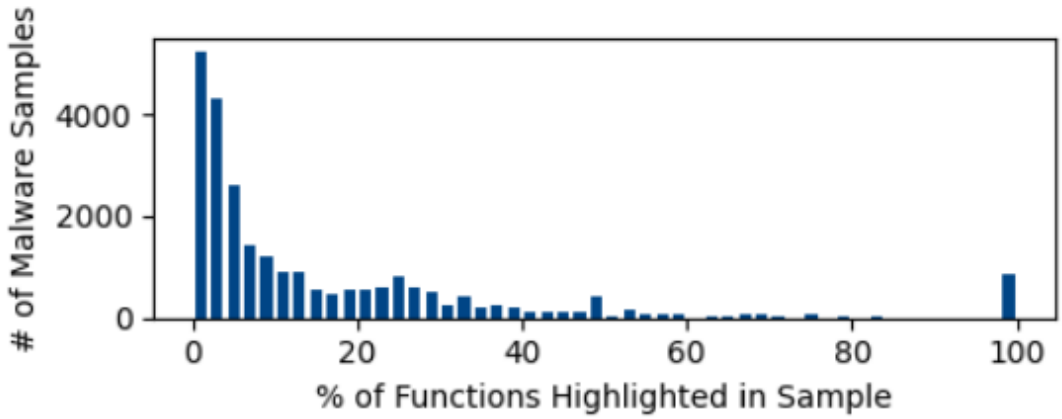

Figure 5: Function Counts. Percentage of functions (per malware sample) the analyst has to review.

	min	max	AVG
Functions	1	527	23.53
	1	26,671	663.81
BB inside f	1	*134,734*	96.02
	1	*22,947*	16.51

False Positives & Prioritization

选阈值时：TPR取40%

	TP	FP	Deepreflect/SHap*
rbot	39	23	0.629/0.487
pegasus	22	80	0.229/0.138
carbanark	8	69	0.111/0.01

*取重构误差排前100的异常component，与SHAP比较precission

False Negatives

	FN	TN
rbot	53	325
pegasus	27	407
carbanark	48	2,111

5.5 Evaluation 4 – Insight

为了评估DeepReflect是否为恶意软件家族间的关系及其行为提供了有意义的见解，作者探索了集群多样性。图4的左侧绘制了 C 中每个类簇中不同家族的数量。由图可知，在家族之间有许多共享的恶意软件技术和变体，部分恶意软件家族间分享了相同的函数，新的恶意软件家族的样本也可以被成功的分类。

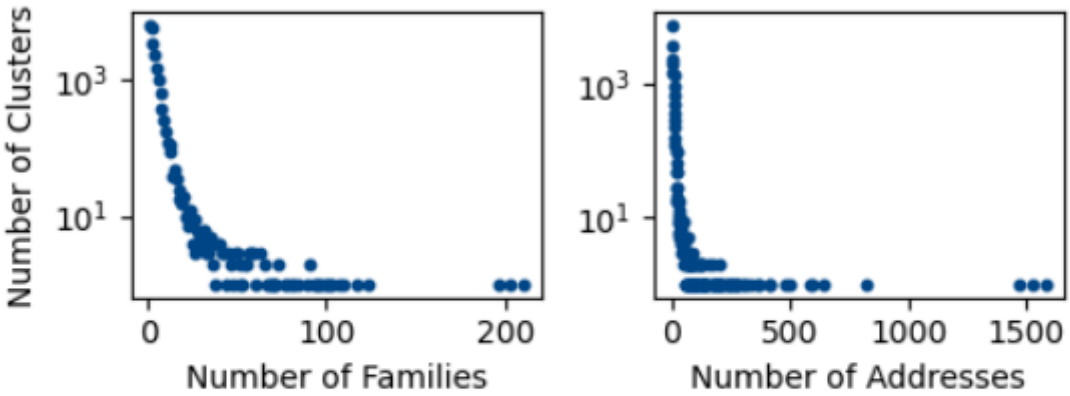

Figure 4: Cluster Diversity. Left: the distribution of families per cluster. Right: the distribution of addresses per cluster to show that there is no bias in function location.

5.6 Evaluation 5 – Robustness

使用LLVM混淆，继续测试模型的鲁棒性；同时使用对抗样本攻击，将包含本文使用的特征的良性样本的代码插入到恶意样本中，但均未对结果产生显著影响。

六、限制和相关工作

本工作的弱点：

- Adversarial Attacks.
- Training Data Quality.
- Human Error.

相关工作：

- Deep Learning and Malware
- Autoencoders and Security

七、总结

在本文中，作者介绍了DEEPREFLECT：一种用于定位和识别恶意组件的恶意软件二进制工具。该工具具有实用性，因为在定位方面不需要标记数据集，而在分类方面只需要一小部分标签，这些标签是分析人员在日常工作流程中逐步收集的。作者希望这个工具和已发布的代码能够帮助世界各地的分析人员，通过确定恶意样本中存在哪些恶意功能以及它们的位置。

八、个人感受

这篇论文介绍了一个名为DeepReflect的工具，它在恶意代码分析领域有着重要的应用。该工具可以定位和识别恶意软件组件，提高了静态逆向工程的效率，并帮助分析人员自动定位和识别恶意行为，以及了解不同恶意软件家族之间的功能关系。同时，这篇论文的论文的写作方式很棒，实验非常完善，工作量也很大，对我来说有一定的理解难度，具体优点如下：

- **评估方面**

  论文的评估部分非常详细和充分。作者进行了对比实验，与企业界和学术界的经典工具进行了比较，如CAPA、SHAP和FunctionSimSearch，这是一种经典的实验比较方式。此外，论文还涵盖了五个方向（Reliability、Cohesiveness、Focus、Insight、Robustness）的详细实验分析，包括附录部分的特征案例和恶意家族行为共享分析，这些内容对于学习都非常有价值，而不仅仅是进行PRF的比较。

- **实战方面**

  论文与ATT&CK框架有效结合，包括映射恶意功能或行为。与ATT&CK框架结合的趋势在安全顶会论文中越来越流行，包括溯源图、APT检测、恶意代码分析、家族分类和二进制等。论文中还详细介绍了各种情景，并配有图形解释，这使观众更容易理解。实验部分的实例对比也非常重要。

- **模型方面**

  论文的模型部分主要使用了半监督学习中的AutoEncoder，可以在样本标注有限的情况下识别更多的恶意行为或类别，减轻了分析人员的手工标注压力。此外，论文还采用了常见的模型方法，如HDBSCAN聚类和PCA降维。整个模型的框架非常出色，并且融合了RoI检测和RoI注释的描述，更容易理解。ROI区域在APP地图热点开发中经常使用，但在二进制领域也能很好地表达，这个跨学科的词汇非常值得关注。

- **特征方面**

  特征方面本文采用4大类（Structural Characteristics、Arithmetic Instructions、Transfer Instructions、API Call Categories）18个特征（之前论文已提出），并且提出了一种解释框架定位恶意软件重要部分的方法，该方法可以映射回原始二进制或控制流图的特征。我认为作者应该思考，在进行恶意代码分析或系统安全研究时，如何尽可能全地覆盖研究问题来提出特征非常重要，并且结合作者的故事。

- **不足之处**

  本文在Limitations部分也有提到，对于Adversarial Attacks，该系统很难抵御这样的攻击。对手可能损害训练集，导致自编码器创建出一个容易隐藏恶意软件功能的易受攻击模型。而且恶意对手可能伪装成benign binary。

  对手还可以通过操纵特征来攻击系统，以阻碍其正常运行。或许这会比较困难，因为这个系统的特征是基于不容易改变的特征。对手必须准确地了解如何修改CFG的结构、指令类型和API调用类型，而又不破坏恶意软件的动态功能。无论是在编译前还是编译后都有可能发生，但我们不能否认这一事件发生的可能性。在未来，可以研制更多反制措施以制止这种Adversarial Attacks。

该系统的自编码器模型严重依赖于良性数据集的内容和质量。自编码器模型严重依赖于良性数据集的内容和质量。如果训练集中缺少某些功能或过多类似恶意的功能，会导致结果的偏差，可能错过关键的恶意功能。

我对解决办法的设想：

1. 对抗性攻击的解决办法可以包括：
    - 强化训练数据集的安全性，确保其不受到污染或篡改。
    - 开发鲁棒性更强的机器学习模型，能够抵御常见的对抗性攻击，例如针对输入数据的扰动或梯度计算的攻击。
    - 实施监测机制，及时检测到对抗性攻击并采取相应的反制措施。
2. 解决训练数据质量的问题可以考虑以下方法：
    - 确保训练数据集的广泛性和多样性，包括各种良性软件和恶意功能。
    - 结合人工智能技术和人工分析，将分析人员的领域知识和经验纳入系统，以帮助纠正训练数据集的不足。
    - 定期更新和扩充训练数据集，以适应新兴的恶意软件和攻击方式。
3. 需要更强的混淆，以及依赖于脱壳质量、可靠的反汇编。