A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis

摘要

1. 本文对基于签名和基于启发式的恶意软件检测方法进行了全面和详细的综述。

- 签名
  - 反病毒软件主要使用基于签名的方法来检测恶意软件
  - 检测 0day 漏洞无效，检测具有混淆技术的软件无效
- 启发式
  - 合法文件被误报为恶意软件
  - 长时间的扫描
- 基于内存的分析
  - 更全面、有前途

1. 文章的几个大方面

- 概述恶意软件类型和恶意软件检测方法
- 讨论当前的恶意软件分析技术及其发现和局限性
- 研究恶意软件混淆、攻击和反分析技术
- 探索恶意软件检测中基于内存的分析结构

绪论

- 为提高效率，使用恶意软件检测系统进行对恶意软件的检测，检测系统包括两个阶段：分析、检测
- 反病毒软件经常通过基于签名的方法检测。
  - 基于签名：提取静态特征，匹配特征库
  - 优点：快速、误报率低
  - 缺点：不能检测未知类型的恶意软件、不能检测具有混淆类型的恶意软件
- 基于启发式(基于行为)的分析
  - 在一个可控隔离环境内运行样本，监控其行为，一旦与已知的恶意软件行为一致，则被标记为恶意软件
  - 优点：能检测未知类型的恶意软件、混淆类型恶意软件
  - 缺点：耗时、误报率高
- 基于内存的分析
  - 转储内存中有大量信息可以用于恶意软件检测

第二部分

A. 恶意软件类型

- 恶意软件：在用户不知情前提下插入系统，损害计算机功能、窃取数据、或逃避访问控制来危害计算机系统。恶意软件常见类别如下： a. 病毒：通过代码注入其他程序来自我复制。可以从程序到程序，从一台电脑到一台电脑 b. 蠕虫：在一台计算机中自我复制，并破坏电脑上的文件和数据。蠕虫可以加密文件、发送垃圾邮件 c. 木马：伪装成合法程序，但执行一些未知的操作，使攻击者可以获得对计算机的控制，获取一些机密

信息 d. 间谍软件：监视用户活动，收集用户信息比如银行卡号、经常访问的网页并发回给攻击者 e. rookit：恶意软件集合，被编程为可以访问计算机并允许其他恶意软件进入计算机 f. 勒索软件：文件加密 g. 广告软件：不停向计算机发广告，通常与一些免费下载的软件绑定，如免费玩游戏 h. 僵尸网络：用于垃圾邮件、拒绝服务攻击

B. 恶意软件检测方法

- 基于签名：分为哈希签名、字节签名
- 基于启发式： 分为动态分析、静态分析

1. 基于签名

- 从恶意软件中提取签名，用签名检测类似的恶意软件。签名是可用于识别特定恶意软件的字节序列或文件哈希。
- 基于签名的方法依赖于实现静态分析来提取称为标记的异常字节序列

局限性

1. 攻击者改变恶意软件签名来逃避被反病毒软件检测并不难
2. 对于使用混淆技术的软件很难检测。混淆技术有以下几种：
    - Dead-Code insertion ： 插入无效代码，使程序外观不同但实际执行的指令相同
    - 寄存器重新分配：切换寄存器或将一个寄存器的值重新分配给未使用的寄存器。
    - 子程序重新排序：子程序是一组执行特定任务的程序操作。这项技术在程序中随机改变子程序的顺序
    - 指令替换：用等效指令替换。如用 PUSH 代替 MOV
    - 代码集成：反编译程序，再将自己的代码插入。是最难的混淆技术

2. 基于启发式

- 基于启发式的也称为异常或基于行为的检测。
- 与基于签名检测的方法不同，基于启发式的检测能够检测未知恶意软件和使用混淆技术的恶意软件。
- 减少了数千个提取的特征，评估他们之间的相似性会影响恶意软件检测
- 基于启发式的通常依赖于数据挖掘技术来理解运行文件的行为，这些技术包括支持向量机、朴素贝叶斯、决策树和随机森林。

C.恶意软件分析方法

- 静态、动态、混合

一、静态分析

- 不运行软件的情况下分析。静态分析恶意软件需要先进行解包，然后使用 IDA 或者 olleydbg 反编译可执行程序
- 检测特征
    - API 调用：API 调用可以揭示程序的行为，可以被认为是恶意软件检测的一个重要标志
    - 字符串签名：字符串揭示了攻击者的意图和目标，因为它们通常包含关键的语义信息。
    - 控制流图：CFG 是一个有向图，它演示程序的控制流，在恶意软件检测中，可以使用 CFG 来捕获 PE 文件的行为并提取程序结构
    - 操作码频率：它标识 CPU 要执行的操作。操作码频率或者操作码序列之间的相似性可以可以作为恶意软件检测的标志
    - 操作序列 n-grams：MALWARE-> "MAL", "ALW", "LWA", "WAR" and "ARE" (3-grams)。应用于 API 序

列和操作码序列
    - 其他特征：文件大小和函数长度，TCP/UDP 端口、目的 IP 和 HTTP 请求等网络特性
- 列举了一些检测方法：表I

TABLE I
SURVEYED PAPERS THAT APPLY STATIC ANALYSIS

Author Year	Static feature	Classifier	Dataset Malware/ Benign	Acc	FP
Hashemi 2018 [23]	Opcode	KNN	M=3,100 B=3,100	91.9%	-
Salehi 2014 [25]	API, arguments	ROT-F, RF, DT, J48, NB	M=826 B=395	98.4%	3%
Han 2012 [26]	APIs sequence	-	M=545	40%*	16%
Santos 2013 [28]	Opcode sequence	DT, KNN, BN, SVM	M=1,000 B=1,000	97.5%	6%
Cheng 2017 [27]	Native APIs sequence	SVM	M=18/ B=72	94.4%	1.4%

* Similarity within the same family

    - Hashemi 和 Hamzeh 提出了一种从可执行文件中提取唯一操作码并将其转换为数字图像的新方法，最后，利用机器学习方法检测恶意软件
    - Shaid 和 Maarof 还建议以图像的形式显示恶意软件。他们的技术捕捉恶意软件的 API 调用，并将它们转换为视觉线索或图像。这些图像被用来识别恶意软件变体
    - Salehi 和 Han 等人都基于提取的API调用构建了他们的技术。Salehi 等人从每个二进制文件中提取 API 调用，并使用API频率来学习分类器。然后，生成“ API 调用列表”、“ API 参数”和“ API 和参数列表”三个特性集，并分别对每个特性集进行测试。
    - Han 等人使用静态分析从 IAT 表(导入地址表)中提取 API。他们将提取的 API 序列与另一个序列进行比较，并计算它们之间的相似性来分类恶意软件家族
    - Cheng 等人也使用 WinDbg 工具分析了本地 API 序列，并应用支持向量机来检测 shell代码恶意软件。

二、动态分析

- 即行为分析。不需要反汇编，比静态更有效，可以检测已知和未知软件
- 模糊和多态恶意软件不能逃避恶意检测

> 多态恶意软件是一种不断变形，演变或改变外观的恶意软件，使得反恶意软件程序很难检测到它

- 缺点：时间密集、资源消耗
- 列举前人的检测方法：表II

TABLE II
SURVEYED PAPERS THAT APPLY DYNAMIC ANALYSIS

Author Year	Dynamic feature	Classifier	Dataset Malware/ Benign	Acc	FP
Liang 2016 [32]	API calls	DT, ANN, SVM	M=12,199	91.3%	-
Mohaisen 2013 [29]	file system, registry, network	SVM, DT, KNN	M=1,980	95%	5%
Mohaisen 2015 [30]	file system, registry, network	SVM, DT, KNN	M=115,000	99%	-
Galal 2017 [33]	APIs sequence	DT, RF, SVM	M=2,000/ B=2,000	97.2%	-
Ki 2015 [34]	APIs sequence	-	M=23,080	99.8%	0%
Fan 2015 [35]	User API, native API	J48, NB, SVM	M=773/ B=253	95.9%	5%

    - 大多数人都是用 API 调用序列作为检测方法
- 恶意软件在受控环境中运行，受控环境及恶意软件对环境的检测方式如下：
    - 模拟器：
        - 一个完全仿真系统控制 CPU，硬盘和资源。仿真器是根据运行环境的受控部分来区分的。
        - 恶意软件仍然可以检测整个环境系统的特征和副作用
    - 调试器：
        - 它是一种观察和检查其他二进制程序执行情况的程序。WinDbg, OllyDbg 和 GDB 都是调试器，可用于在指令级别监视可疑二进制文件的执行行为
        - 恶意软件会在系统上安装调试工具的迹象，例如搜索注册表项、文件和目录，一旦有就停止执行
    - 沙箱仿真器：
        - 模拟操作的程序，以供用户观察而不实际执行操作，如 CWSandbox, Norman sandbox 和 Detours 允许恶意软件在一个受控的虚拟环境中执行并记录其行为
        - 恶意软件检测文件系统、注册表、进程信息等判断是否在沙箱中运行
    - 虚拟机：(最普遍)
        - 懂得都懂
        - 文件系统、注册表和进程列表中搜索安装 VM 工具的工件来检查系统上是否存在虚拟机；以及某些可以由用户模式调用的指令，如 “sidt”、“sgdt”和“sldt” ，以观察虚拟机工具的存在；一些硬件特点也可以用于检测，如在真实系统中，CPUID 管理程序位被设置为零，因此恶意软件可以测试这个位，以确定它们是否在虚拟机中运行

三、混合分析

- 混合分析综合静态分析和动态分析的优点。
    - 静态分析：廉价、快速和安全。然而，恶意软件通过使用混淆技术来逃避它。
    - 动态分析：可靠的，可以识别混淆。此外，它能够识别恶意软件变体和未知的恶意软件家族。然而，它是时间密集和资源消耗的
- 提出的混合分析的算法：表III

TABLE III
SURVEYED PAPERS THAT APPLY HYBRID ANALYSIS

Author Year	Feature Static/ Dynamic	Classifier	Dataset Malware/ Benign	Acc	FP
Shijo 2016 [43]	PSI/ API calls	RF, SVM	M=1,368 B=456	98.7%	-
Islam 2013 [44]	Function length, PSI/ API	DT, SVM, RF, IB1	M=2,939	97%	5.1%
Ma 2016 [45]	Import functions/ call functions	DT, NB, SVM	M=279	-	-
Santos 2013 [46]	Opcode/ system calls, operations, and exceptions	DT, KNN, NB, SVM	M=13,189 B=13,000	96.6%	3%

    - Shijo 和 Salim 提出了一种综合技术来检测和分类未知文件，通过静态分析提取可打印字符串信息，通过动态分析提取 API 调用
    - Islam 等人从静态分析中提取了两个特征函数长度频率(FLF)和可打印字符串信息(PSI)以及动态分析中的API调用和参数，并使用随机森林分类
    - Ma 等人在恶意软件分类中引入了一种减少误报的方法，称为集成，将静态和动态分类器结合到一个分类器中。该方法采用了静态导入函数和动态调用函数等多种特征，提高了准确率，减少了误报。

- Santos 等人引入了 OPEM ，这是一种通过将静态分析中获得的操作码频率与动态分析中的系统调用、操作和引发异常相结合来检测未知恶意文件的工具

四、内存分析

- 对恶意软件进行全面的分析，检查正常范围之外的函数 hook 和代码
- 内存取证分为两个阶段：
  - 内存获取：通过 Memoryze、FastDump 和 DumpIt 等工具转储目标机器的内存以获取内存映像
  - 内存分析：使用 Volatility 和 Rekall 等工具分析内存映像，查找恶意活动。
- 前人提出的内存分析算法：表IV

TABLE IV
SURVEYED PAPERS THAT APPLY MEMORY ANALYSIS

Author Year	Feature	Classifier	Dataset Malware/ Benign	Acc	FP
Mosli 2016 [3]	Registry, imported libraries, API calls	SVM, SGD, DT, RF, KNN	M=400 B=100	96%	-
Mosli 2017 [52]	Number of opened handles	KNN, SVM, RF	M=3,130 B=1,157	91.4%	-
Zaki 2014 [7]	Driver, module, hooks, callback	-	-	-	-

  - Teller 和 Hayon 提出了基于以下事件触发内存转储:基于 API、基于性能和基于仪器，以便了解在执行恶意软件文件期间发生了什么，而不仅仅是在它结束时。
  - Choi 等人在 Teller 和 Hayon 的技术进行了修改。通过为 Cuckoo 沙箱实现基于API 触发器的内存转储技术，使得 Cuckoo 能够在每次需要的 API 调用时转储内存。
  - Mosli 等人调查了从内存映像中提取的三个特征:导入库、注册表活动和用于检测恶意软件的 API 函数调用。使用支持向量机和注册表活动数据，其最高准确率约为 96%。
  - Zaki 和 Humphrey 研究了内核级 rootkit 留下的内存工件，例如:驱动程序、模块、SSDT 钩子、IDT 钩子和回调。实验已经证明回调函数、修改的驱动程序和附加的设备是内核级最可疑的活动。
- 内存取证可以监控各种恶意软件行为。恶意软件行为如下：

1. API hook
   - IAT hook:API 函数的地址存储在IAT (Import address Table)中。恶意软件会覆盖 IAT 中 API 的位置，从而迫使进程调用攻击者函数而不是原来的 API
   - Inline hook:JMP 指令后修改为自己的函数地址
   - IDT hook:中断描述符表(IDT)存储处理中断和异常的函数。恶意软件改变 IDT 中0x2E条目中的值，并在执行对内核模式 API 函数的调用时获得控制。
   - SSDT hook:系统服务描述符表(SSDT)是一个包含指向内核模式函数的指针的表。恶意软件使用 SSDT 来保护和隐藏自己。
   - IRP hook: Windows 中的应用程序通过 IRP (Input/ Output Request Packet)与驱动程序通信。恶意软件使用 IRP 钩子进行恶意操作，如键盘记录和磁盘访问过滤。
2. DLL 注入：将恶意代码插入到合法进程中的技术。一旦恶意 DLL 被注入，执行流就被转移到恶意内存空间
   - 经典的 DLL 注入：使用远程线程。恶意软件在目标进程的虚拟地址空间中写入动态链接库(DLL)路径，并在目标进程中创建远程线程，以确保装入恶意 DLL。
   - 通过修改注册表的注入：修改特定表项值，使进程加载恶意软件库
   - 使用窗口钩子函数注入：恶意软件可以在特定事件(如鼠标移动或者按键)被触发时加载它们的恶意 DLL
3. 进程隐藏
   - 使用隐藏 rookit，可以修改二进制程序

    - 使用直接内核对象操作(DKOM)修改内核数据结构，隐藏进程

      > 在Windows操作系统中，系统会为每一个活动进程创建一个进程对象 EPROCESS，为进程中的每一个线程创建一个线程对象 ETHREAD。在 EPROCESS 进程结构中有个双向链表 LIST_ENTRY，LIST_ENTRY 结构中有 FLINK 和 BLINK 两个成员指针，分别指向当前进程的前驱进程和后继进程。如果要隐藏当前进程，只需把当前进程的前驱进程的 BLINK 修改为当前进程的 BLINK，再把当前进程的后继进程的 FLINK 修改为当前进程的 FLINK。

- 反存证技术如下：
    - 内存隐藏
    - 内存获取失败：检测特定进程或者限制驱动加载阻止内存获取
    - Anti-Carving：恶意软件试图通过操纵内核数据结构字段阻止内存分析工具从内存中提取有效信息
    - 增加时间：恶意软件在内核数据中制造一些无用的对象以增加分析时间

第三部分

A. 未来的方向

- 恶意软件的挑战
    - 生成恶意软件变体的自动化。攻击者可以开发自动化工具，每天能够生成数千个不同的恶意软件样本
    - 恶意软件团队可能会提供这些恶意软件自动化工具来出租或出售，给低技能团队和业余黑客进入恶意软件世界的机会
    - 恶意软件在结构和功能方面变化迅速，现有恶意软件检测技术的检出率很难保证
    - 恶意软件可能使用新的加密方法或混淆技术，使检测更加困难
- 内存检测信息多，比较有前途

B. 数据集

- 可以使用蜜罐收集样本
- 也可以从一些知名网站下载，如Malware DB, Malwr, MalShare, VX Heaven, theZoo 和VirusShare
- 专业公司或项目组会分享收集的恶意软件数据集

第四部分：结论

- 恶意软件开发者开发复杂的软件：频繁改变签名、使用新的混淆技术逃避检测
- 总结全文:
    - 恶意软件类型和恶意软件检测方法
    - 回顾三种恶意软件分析技术：静态、动态和混合
    - 内存分析在恶意软件检测中的方法
    - 研究恶意软件未来发展方向和恶意软件数据集的来源

Android malware detection & protection: a survey

> McLaughlin N, Martinez del Rincon J, Kang B J, et al. Deep android malware detection[C]//Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. 2017: 301-308.

> Keywords： Malware Detection, Android, Deep Learning

参考链接 http://git.a101e.lab:20080/chencwx/papers/-/blob/master/%E8%AE%BA%E6%96%87%E9%98%85%E8%AF%BB/42-Deep%20android%20malware%20detection/42-Notes.md

Summary

- 这篇论文使用二进制程序反编译的静态操作码作为该程序的特征，使用`CNN（convolutional neural network）`为模型，取得了较好的效果，使用`accuracy`、`precision`、`recall`、`f-score`等评价标准来评价该模型。在本文的数据集实验环境下，F-score=0.97，在实际的数据集环境中，F-score=0.71。 虽然实际环境中的效果不是特别好，但是模型训练所需要的时间相对较少，效率较高，而且内存使用率是一个常数。
- 它不需要进行动态特征提取，框架更加简单
- 比n-gram相比，效率提高了很多，虽然真正的效果一般，但是具有创新性，对论文的撰写来说，提供了一种「尝试将已有的处理其他方面的机器学习算法用于我们想要实现的目标上」的思路，说不定会获得不错的效果

Glossary

- android malware:基于安卓平台的恶意软件
- CNN：由3个部分构成
  - 卷积层：提取局部特征(filters和激活函数构成)
  - 池化层：大幅降低参数量级(降维)
  - 全连接层：类似于传统神经网络的部分，用来输出想要的结果
- opcode：操作码，机器指令中的指令部分

Research Objective(s)

- apply convolutional neural networks to the problem of malware detection.（将CNN应用于安卓恶意软件的检测）

Background / Problem Statement

- many of these methods are reliant on expert analysis to design the discriminative features that are passed to the machine learning system used to make the final classification decision. --当前大部分的恶意软件检测的方法依赖于特征的选择和设计
- Recently, convolutional networks have been shown to perform well on a variety of tasks related to natural language processing -- CNN在自然语言处理方面表现良好
- 考虑选择静态特征-操作码，并结合CNN进行训练

Method(s)

Disassembly of Android Application

- 特征提取

> 先将apk文件反编译，得到apk的所有资源文件(其中的.smali文件内的语法使用Dalvik虚拟机指令语言)。 之后，从.smali文件中提取每个method的操作码序列，组合得到当前smali文件(一个class)的特征数据集。 最后，所有的smali文件提取的操作码序列汇集得到单个apk文件的操作码序列。

Network Architecture

- 整体框架图

Opcode Embedding Layer

- one-hot编码操作码序列，X[n]是第n个向量表示的操作码；通过查看Dalvik文档，向量的维数(dimension)是218维。 X[n]乘一个权重矩阵W(随机初始化)后，将操作码映射到一个k维的Embedding空间中，得到P[i]矩阵。

> 这样做的目的是：让网络可以在一个连续的k维空间中学习到每个操作码的正确表示。 Embedding空间的维数会影响网络表示映射关系的能力，因此，使用更高的维度可以使网络具有更强的灵活性去学习高维的非线性映射关系。

Convolutional Layers

- 第一个卷积层接受`N*K`的Embedding矩阵作为输入，更深层次的卷积层以之前卷积层的输出作为输入。 每个卷积层有M[l]个filter(第一层size是`s[1]*k`，更深层的size是`s[l]*M[l-1]`)，这意味着第一层的filter可以检测s[1]个操作码。 在样例前向传播的时候，每个filter都会产生一个激活映射Al,m，所有的filter产生的组合在一起，构成矩阵Al 第一个卷积层的filter产生的矩阵P可以用下面的第一个公式表示:
- 在更深层次的卷积层中，使用A[l-1]作为l层的输入
- 经过最后的卷积层后，接着使用下面的公式进行maxpooling，使得f的长度是M[l]（本文中的M[l]值为2）。

Classification Layers

- 经过神经网络训练以后输出`z`
- 使用`softmax`（归一化函数）得出分类
- 卷积层的结果向量f，作为多层感知机(MLP)的输入-一个隐藏层和一个输出层。 其中隐藏层使用Relu作为激活函数，最后输出Softmax归一化的概率值。

Learning process

- 训练的损失函数如下
- 梯度下降方式如下，同时梯度的权重和样本的数量成反比，防止出现由于某类样本的数量过多导致检测结果偏向这类的结果,其中a是学习率，为了使损失函数达到最小值，达到最好的训练效果
- 权重如下

```
if num(malware)=M,num(benign)=B,M<B
weight(malware)=1-M/(M+B)
weight(benign)=M/(M+B)
```

Evaluation

datasets

- Small Dataset: Android Malware Genome project - 863 benign & 1260 malware
- Large Dataset: McAfee Labs - 3627 benign & 2475 malware
- V. Large Dataset: Mcfee Labs - 9268 benign & 9902 malware

results

- full results in datasets aforesaid
- Realistic Testing results
- Realistic Testing results
- Learning Curves

> 学习曲线，1-F-score 和训练样本的关系

Conclusion

- 提出了一种新颖的 Android 恶意软件基于深度神经网络的检测系统，并已在四个不同的 Android 恶意软件数据集中进行了验证，发现有良好的效果。
- 能够同时训练大量特征，且仅需给出大量标记恶意软件样本的操作码序列。
- 不再需要人工设计特征提取，且比现有的基于 n-gram 的计算效率更高，并且可以实现在GPU上运行。

Deep android malware detection

文献来源

背景

主要内容

创新性

A comparison of static, dynamic, and hybrid analysis for malware detection

文献来源

背景

主要内容

创新性

Robust intelligent malware detection using deep learning

文献来源

背景

主要内容

创新性