

## 使用树形图和线程图对恶意软件行为进行可视化分析

非官方中文译本 · 安天实验室译注

文档信息			
论文题目	Visual Analysis of Malware Behavior Using Treemaps and Thread Graphs		
论文作者	Philipp Trinius、Thorsten Holzy、Jan Gobel、Felix C. Freiling		
发布单位	德国曼海姆大学可靠性分布式系统实验室		
原文链接/出处	<a href="http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&amp;arnumber=5375540">http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&amp;arnumber=5375540</a>		
论文发布日期	单击此处输入日期。	译文发布日期	2014/10/8
论文摘要&关键词	<p><b>摘要：</b>本文研究了恶意软件行为的可视化，目标是帮助分析人员快速获取新恶意软件样本的本质，并对其进行分类。我们的技术基于沙箱环境自动生成的详细的行为报告。本文还探究了两种可视化技术：树形图和线程图。我们认为两种技术都可以有效地支持分析人员对软件的恶意性进行检测并对恶意行为进行分类。</p> <p><b>关键词：</b>进攻性软件；信息可视化；行为分析</p>		
译者	安天技术公益翻译组	校对者	安天技术公益翻译组
免责声明	本译文为安天实验室针对网络资料翻译而成，并未取得原作者授权，仅供内部学习和交流使用，安天实验室不对任何可能因此导致的版权问题承担责任。		

# 使用树形图和线程图对恶意软件行为进行可视化分析

Philipp Trinius , horsten Holz , Jan Gobelz, Felix C. Freiling

德国曼海姆大学可靠性分布式系统实验室

**摘要：**本文研究了恶意软件行为的可视化，目标是帮助分析人员快速获取新恶意软件样本的本质，并对其进行分类。我们的技术基于沙箱环境自动生成的详细的行为报告。本文还探究了两种可视化技术：树形图 ( treemap ) 和线程图 ( thread graph )。我们认为两种技术都可以有效地支持分析人员对软件的恶意性进行检测并对恶意行为进行分类。

**关键词：**进攻性软件；信息可视化；行为分析

## 1. 简介

恶意软件是最活跃也是最具威胁的存在。近年来，恶意样本的数量大规模增长。对新的工具和技术进行开发，让恶意样本的分析不受限于人力，这是我们义不容辞的责任。

分析恶意软件是一项十分琐碎的任务，因为攻击者使用了代码混淆技术。现有的分析技术无外乎静态和动态分析两种方式。静态分析就是对样本的代码进行分析。静态分析的优势是可以获取已知恶意软件的整体结构。然而，静态分析耗时耗力，而且很多攻击行为都采用躲避静态攻击的技术。静态分析还无法保证分析的自动化。

动态分析就是分析恶意软件的行为。沙箱技术的开发保证了对恶意软件分析的动态性。恶意软件样本在被控制的环境中执行，其对文件系统或注册表的修改都可以被记录下来。与静态分析相比较，动态分析保证了高水平的自动化程度。

沙箱报告，通常包含数百个条目，可以提供详细的分析结果，而手工分析只能侧重于某一特定样本的快速捕获。假设许多样本只是已知恶意软件家族的一小部分变种，那就很有必要让分析人员更快速更轻松地掌握恶意软件样本的行为。本文就将这种设想的抽象化和可视化进行了有效的结合。

本文的主要贡献：

首先，我们呈现了一种将沙箱报告抽象成概要的参数化方法。然后我们利用分类技术将导出的结果进行了可视化和可读化。我们认为两种技术都可以对恶意软件的主要特征提供独特的视角，这样也更好地支撑了人们对恶意行为的检测。而且，我们认为这种可

视化的呈现也让分析人员对恶意软件样本进行分类提供了快速的途径。

我们使用的可视化技术主要有两种：树形图 ( treemap ) 和线程图 ( thread graph )。树形图展示了某一样本行为的分布情况。树形图将这些信息以一种巢状的矩形呈现出来，以便更好地掌握样本的整体行为。由于树形图没有展示出恶意行为发生的顺序，我们利用线程图将某一样本的线程进行了可视化。分析人员可以据此了解有关每个单一线程行为的信息。

我们通过展示一些案例研究来证明这种技术的可行性。首先，通过分析四种不同的恶意软件类，我们证明了我们的方法可以用于寻找属于同一恶意软件家族的样本。其次，通过分析一些存在恶意软件的数据文件，比如利用了 Acrobat Reader 漏洞的 pdf 文件，我们证明了我们的技术还可以用于检测恶意行为。

## 相关研究

Xia 对程序流数据的可视化研究与我们当前研究的目的是相同的，那就是将给定恶意软件样本的行为可视化。Xia 的研究侧重于数据传递和入侵跟踪，而我们使用了一种简单的抽象技术将分析人员所观察到的行为进行了可视化。我们的线程图与 Xia 的研究更接近，但却包含了更多的有关每个线程实际行为的信息。当前还没有任何其他使用树形图对行为信息进行可视化的研究。

Conti 在其论文中介绍了将二进制和数据文件可视化并进行逆向工程的不同技术。其目的也是为了更好地帮助分析人员，然而 Conti 使用的是静态的分析技术。由于恶意软件一般被可执行的加壳器封装过，单个字节的传播是十分相似的，并且其结构信息也都丢失了，所以在应对恶意软件时，Conti 开发的 *byteview* 和 *byte presence* 可视化技术在应用时存在一定的困难。

文本的研究框架如下：第二节对系统进行了综述，第三节基于参数式的抽象方式对恶意软件样本的行为进行了可视化研究，第四节对具体的恶意软件可视化样本进行了描述，第五节介绍了可视恶意软件的分类技术，第六节展示了非可执行文件的可视化，第七节进行了总结和展望。

## 2. 系统设计

图 1 展示了我们系统的综合框架。我们利用蜜罐收集了大量的恶意软件样本。为了分析这些样本，我们首先将这些样本在沙箱中执行，并观察它们在运行过程中的行为。第二步，我们使用不同的技术将收集到的信息进行可视化处理。最后，我们的技术生成了能够反映样本行为的不同的可视化图景，从而帮助分析人员快速判断给定恶意样本的行为。



图 1：恶意软件行为可视化系统的框架

动态分析的优势是它在部分程度上规避了代码混淆（比如加壳器或加密器）所带来的问题。由于软件知道如何对自身进行解包，我们只需要将其执行，让样本自己解包或解密，然后再研究它的行为。而且，这一过程可以实现很高的自动化水平。在我们的研究中，我们选择了沙箱环境来进行动态分析。在执行过程中，沙箱观测到了系统级别的改变，比如系统文件或 Windows 注册表的改变，发送或接收到的网络包。沙箱输出的结果就是对观测到的行为的总结报告。

在系统中，我们使用的是以 CWSandbox 为基础的行为分析报告。我们将用于分析的样本在 CWSandbox 中运行两分钟，让工具来记录所有系统级别的活动。分析结果会以 XML 的格式输出，内容包括每个线程被观察到的行为，比如有关所有被加载的系统注册表的信息，输入以及输出的网络连接，被窃取权限的注册表项，以及许多其他事件。每天大概可以生成 2500 到 4000 份报告，我们因此获得了数量可观的，详细描述了样本线程执行情况的报告。

由于基于行为的分析方式可以对可执行以及数据文件进行分析。对于数据文件，可以将其在相关的设备上开启，并观察其行为，从而实现对数据文件的间接分析。例如，给定一个 pdf 文件，我们可以将其在 Acrobat 阅读器上开启，然后分析阅读器的行为。我们在后文中将展示这种方式可以用于检测数据文件的异常行为。

## 3. 基于抽象的可视化

每个沙箱报告都相当长（比如，XML 格式，数千

字节），因为沙箱分析了许多 API 函数，记录了所有相关的活动。有效的可视化操作是需要从这些报告中提炼或抽象出相关信息的。因此，我们将给定的 XML 报告转换成更易于展示的格式。我们将每个 API 函数根据其功能的相似性与否划分为特定的单元，例如，所有与文件系统活动相关的 API 函数都属于一个单元。而且，我们还将每个 API 函数的参数根据相关性进行了排序，比如更加显著的参数被列在前面，不够显著的参数则被完全省略了。

### 3.1 抽象程度

沙箱报告可以生成四种不同的抽象水平。

- 为了对某一进程进行大致的了解，我们仅展示该进程的某些单独的部分，比如，能够显示该样本是否获取了注册表权限的部分。这是 Level 1。
- Level 2 与 Level 1 相比，多了每个部分中被执行的 API 函数的名称。
- Level 3 则添加了有关每个 API 函数最显著参数的信息。
- Level 4 添加了更多的参数和细节。

在对恶意软件行为进行可视化时，Level 2 和 Level 3 已经足够用了。

### 3.2 可视化

我们采用了两种可视化途径，每种都可以帮助分析人员快速了解给定样本的行为。树形图为恶意软件样本的行为以及发生频率提供了综合图景，线程图则将某一进程的线程行为进行了可视化。

#### 3.2.1 树形图

第一种可视化途径中，我们使用树形图技术将行为报告转化成标准化格式。树形图展示的信息是一组巢式矩形，我们需要定义一种铺瓦式的算法来框定树形图的结构。每个矩形的宽度与行为报告中属于某一单元的 API 函数的百分比是成正比的，也就是说，属于某一单元的 API 函数越多，相应的矩形则越宽。我们一共观察了 20 个单元的 120 多个 API 函数。树形图中每个单元都有固定的顺序和颜色。最左边的红色单元叫做 com 单元，包含三个 API 函数。蓝色部分叫做 dll handling 单元。在某些报告中，并不是所有单元都有 API 函数，则相应的树形图中也未显示出该单元。为了解析树形图中包含的信息，位置和颜色都需要考虑在内。图 2 中展示了恶意软件样本 *Adultbrowser* 的



树形图。我们可以看到其中两个单元明显比其他单元要宽，表明该样本进行了更多的特定操作。为给定单元绘制的矩形高度也可以反映该单元内操作进行的频率。

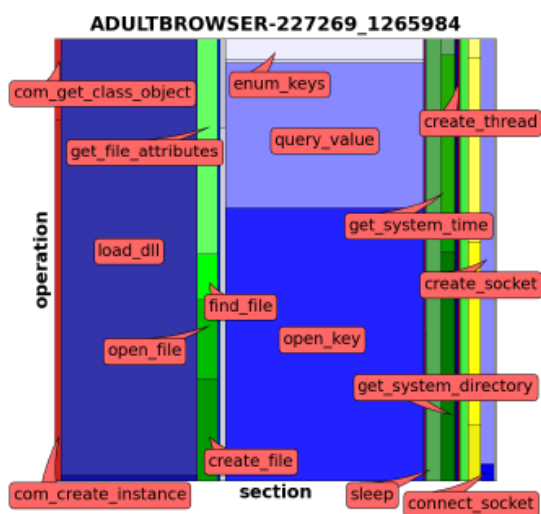


图 2：恶意软件 Adulthood 的可视化树形图

### 3.2.2 线程图

在第二种途径中，我们将给定样本的行为按照时间顺序进行了可视化。我们绘制了一个能够代表系统命令被执行的时间顺序以及二进制文件所生成的不同线程的图表。x 轴代表时间，y 轴代表操作。我们将这种绘图结构称之为线程图。

由于一个线程可能包含数千项操作，因此需要设置一个阈值来获得清晰、连贯的可视化效果。因此，线程图中仅展示了最佳数量的操作行为。在我们的试验中，550 这个数目既可以保证准确性又可以保证清晰性。

图 3 中展示的同样是 Adulthood 恶意样本，但这次用的是线程图。我们可以看到用一个线程就可以表示该样本的绝大多数操作行为。这一线程展示了该样本的许多注册表操作以及最初的网络和系统相关操作。此外，还有两个额外的线程，但在整个分析过程中，其操作次数比较有限。

这些线程图的绘制是依据沙箱中该恶意软件样本执行的时间长度，代表了每个恶意软件家族的独特行为。这意味着任何其他能够生成同样图像的样本都可以被归类为某种已知恶意软件。我们将在下一节中展示更多的案例研究。

## 4. Hookshell 的可视化

我们选择了一个恶意软件样本并详细研究了它的可视化行为。我们与 2009 年 4 月 29 日分析了 MD5 值

为 a9f6aa1649e6a0f1bfad8a576f0193a0 的文件，反病毒引擎，如 Antivir、SecureWeb-Gateway 和 Sunbelt 等将其归类为 Hookshell 恶意软件。CWSandbox 报告内容很少，仅包含描述其行为的 379 行内容。

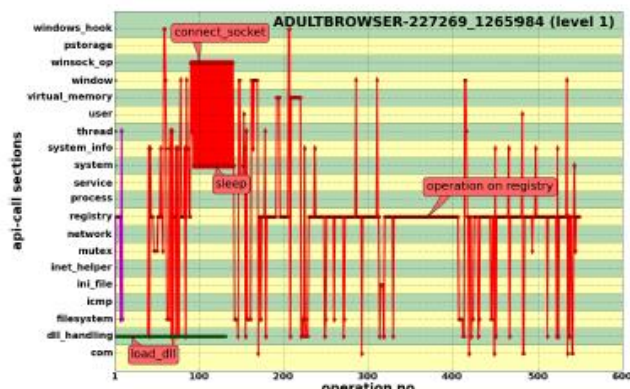


图 3：使用线程图生成的 Adulthood 恶意软件的图像

图 4 展示了 CWSandbox 报告生成的树形图。从图像中我们可以清楚地看到，该恶意软件执行了来自六个不同单元的 API 函数，从左到右分别是 dll handling 单元、filesystem 单元、inifile 单元、registry 单元、process 单元以及 system info 单元。所有单元的水平分支显示，每个单元至少有两种不同的操作被执行。

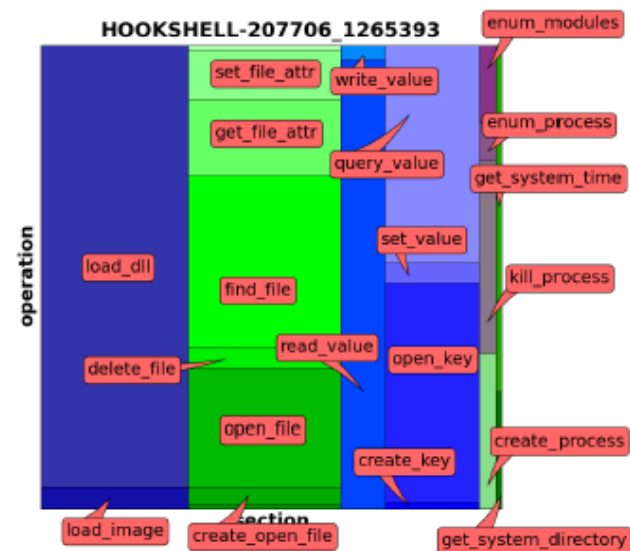


图 4：Hookshell 恶意软件样本的树形图

如果想要对恶意软件样本的行为进行详细的分析，我们则需要考虑下图 5 和图 6 中展示的线程图。Level 1 的线程图表明恶意软件执行了六个单元的操作。每个线程可以通过其操作行为而得到识别。从线程图中可以判断出四种运行不同 API 函数的线程。其中两种在执行 API 函数的顺序一模一样，因此它们的颜色变黑了。为了回避两种或更多线程的重叠，可以为每个线程绘制一张图像。

通过仔细研究图 6 中 Level 2 的线程，我们可以将

有关恶意软件行为的详细信息提取出来。所有线程都在初始阶段加载了一组程序库，用最开始的黑色直线代表。红色的线程表示文件系统改变了某些文件属性并创建了一个新文件。随后，它还读取了某些注册表项，并且频繁地查询一个 ini 文件，最终创建了一个新的注册表项，而其他线程并未在文件系统中有所操作。从图像中无法清楚地判断该样本读取并创建了哪些文件和注册表项，具体的信息需要从沙箱报告中提取。

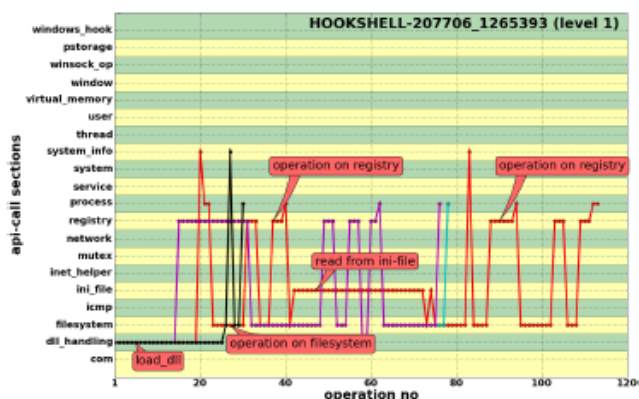


图 5：Hookshell 恶意软件样本的 Level 1 线程报告

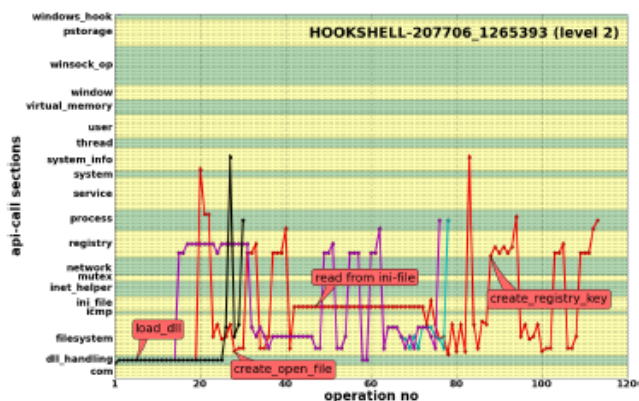


图 6：Hookshell 恶意软件样本的 Level 2 线程报告

## 5. 可视化恶意软件的聚类

我们在本节展示了如何将可视化的恶意软件行为报告用于判断给定样本是否属于某一恶意软件家族。这种聚类方法可以帮助分析人员决定是否对某一新样本进行更深入的分析或判断该样本是否属于某一已知恶意软件类别。

作为测试用的样本组包含 2000 个恶意软件样本，它们实现已经经过六种反病毒软件进行了分类和标注。因此，每个图像的名称中都体现了它的分类信息。我们一共有 13 种不同的标签，它们分别是：Adultbrowser, Allapple, Bagle, Bancos, Hookshell, Looper, Podnuha, Porndialer, Rotator, Sality, Spybanker, Sramler 和 Swizzor。

因此，每个分析结果中树形图被分成 13 组，每组展示各自的图像。对于大多数检测的样本，这种情况

都是可行的。我们用 CWSandbox 分析了恶意软件样本，并将观察到的行为进行了可视化。图 7 展示了四个恶意软件类的八个树形图。不同恶意软件样本的可视化图像都是不同的，但是同样恶意软件的不同样本却是几乎相同的。然而，这并不适用于所有情况，因为对样本行为进行分析的结果可能会受到一些不可控因素的影响，比如，无法连接到某一僵尸网络的命令与控制服务器，或者该僵尸网络呈现出完全不同的行为。

## 6. 非可执行文件的可视化

我们的可视化途径不受限于可执行文件。在沙箱中用相关设备开启一个数据文件就可以生成相应格式的行为报告。通过运行 pdf 阅读器并在沙箱中执行我们想要分析的文件，就可以将 pdf 文件的行为可视化。在本节中，我们将利用良性的和恶意的 pdf 文档简要解释下间接行为分析和可视化的概念。

我们使用了 17 个恶意 pdf 文件以及 200 个通过 Google 搜索随机选择的良性 pdf 文件。我们用 Avira 反病毒引擎对这些文件进行了扫描。200 个 pdf 均被标记为安全文件。17 个被标记为恶意的 pdf 文件中，有 15 个在我们的沙箱环境中表现出了异常的行为，因为这些文件都尝试进行网络连接。我们使用了 Adobe Acrobat Reader 8.0 阅读了这些文件，人工确认了这些文件的恶意性。

依据恶意软件作者为软件架构的环境以及嵌入的检测模块，可执行文件或数据文件的恶意软件可能被执行也可能不被执行。我们研究的案例中，恶意软件作者可能会检测系统中是否安装了带有漏洞的 Adobe Acrobat 阅读器。

间接行为分析的结构显示，Acrobat 阅读器对所有良性数据文件的行为不不一致。基于它们的可视化行为，我们将其分成两类。图 8 (a) 和 (b) 展示了每一类的树形图。图像中最左侧的操作可以将它们区分开，在图 (a) 显示为红色，在图 (b) 中则消失了。这一额外操作表示的是 API 函数 com section。这种特性在相应的线程图中也可以观察到。是否执行函数与 pdf 文件中的定义有关，与该行为是否可疑或恶意并无关系。

用恶意 pdf 文件的分析报告生成的可视化图像则不尽相同。因为行为报告中还包含被分析的恶意文件初始化的额外进程操作。这些执行不同操作的进程属于恶意 pdf 文件中被植入的命令所下载的二进制文件。

图 8 (c) (d) 展示了两种恶意 pdf 文件的树形图。即便如此，乍一看去两个树形图看起来还是很相似的。恶意 pdf 文件和良性 pdf 文件有三种区别，它们分别是：

1. 恶意 pdf 文件的紫罗兰/蓝色部分有轻微的延伸趋势。

2. 恶意 pdf 文件的树形图有额外的亮绿色部分延伸到左侧的黄色部分。
3. 只有恶意 pdf 文件的树形图的右侧黄色部分较宽。

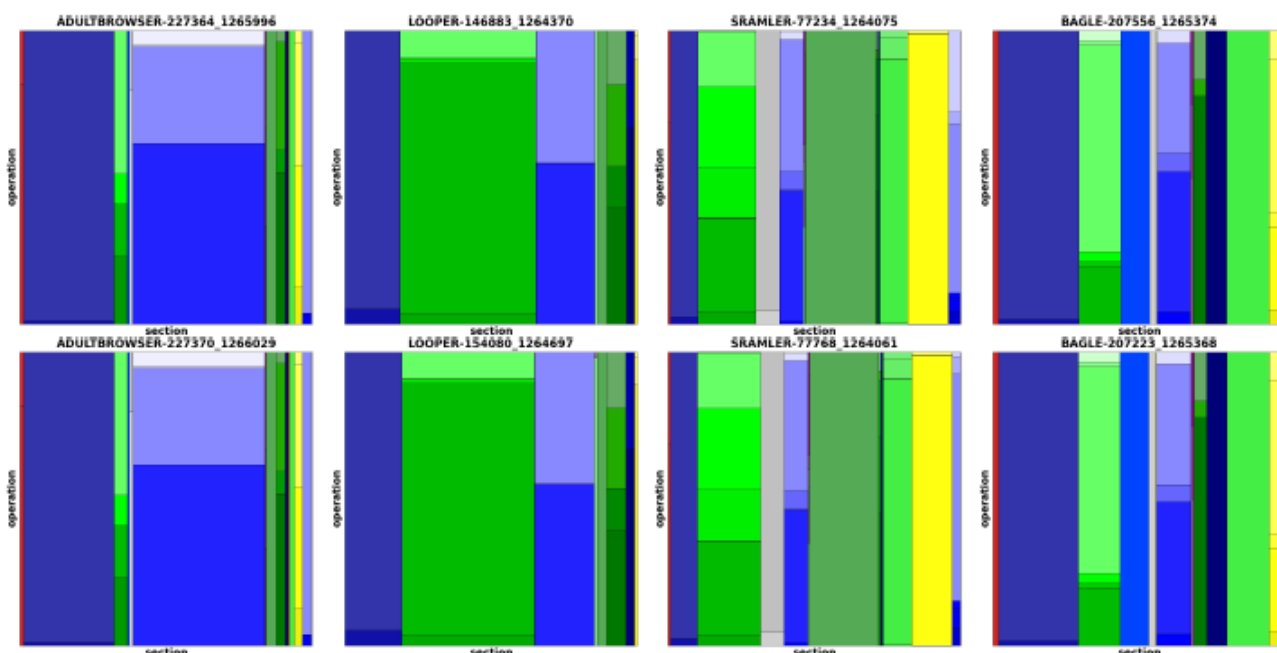


图 7：四种不同恶意软件的树形图

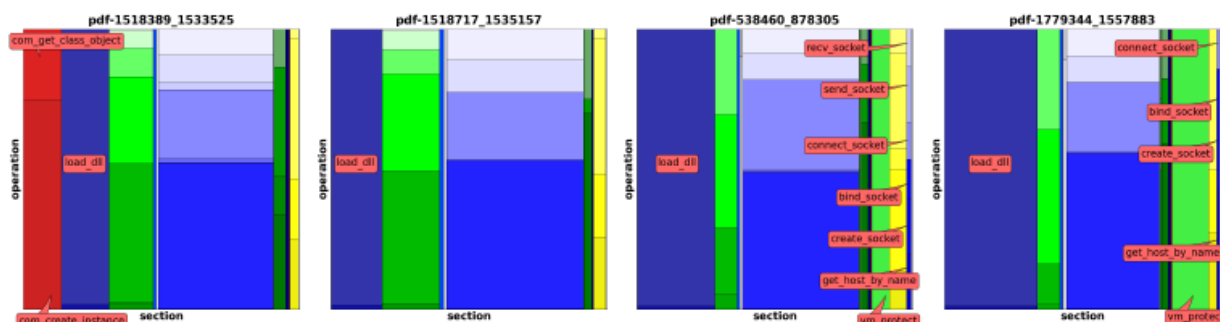


图 8：良性 pdf 文件的 (a) - (b) 树形图、恶意 pdf 文件的 (c) - (d) 树形图

通过比较良性和恶意 pdf 文件的树形图，我们就可以对其进行区分了。接下来我们将研究造成两者间差异的 API 函数。

树形图中紫罗兰/蓝色部分代表 dll handling 单元，该单元包含三个 API 函数 load image load dll 和 get proc address。dll 文件可以为调用的程序提供有用的功能。每个微软系统的软件在启动时都会加载一些 dll 文件。这一行为在图 9 的线程图中也显示出来了。在良性 pdf 线程图中，一条红线表示 dll 文件的加载操作。恶意文件的线程图中出现了第二条绿线，表明另一个线程加载了更多的程序库。加载更多程序库的操作就可以帮助我们判断其属性，因为所需 dll 文件的数量仅仅在于设备而非它所开启的文件。

接下来我们要看一下黄色部分左侧的亮绿色部分。

这部分代表 CWSandbox 报告的 virtual memory 单元，其中包含需要分配、读取和写入虚拟内存中的 API 函数。恶意的 pdf 文件需要额外的内存来保存变量和代码，一般都是与漏洞相关的代码。由于良性 pdf 文件在 Acrobat 阅读器中运行，它们不需要此类内存分配。

第三个特点就是黄色部分右侧的额外部分。尽管这部分很难观察到，但对于区分良性和恶性性来说却是最重要的。这部分被叫做 winsock operation 单元，其中包含所有与网络相关的 API 函数。在图 9 (c) 和 (d) 中，有两个额外的线程—紫罗兰色和黄色，它们执行的操作就属于这一单元。另有一些操作会将恶意软件下载到存在漏洞的系统中。开启网络连接下载额外恶意软件的 pdf 文件就可以被判定为具有恶性性。



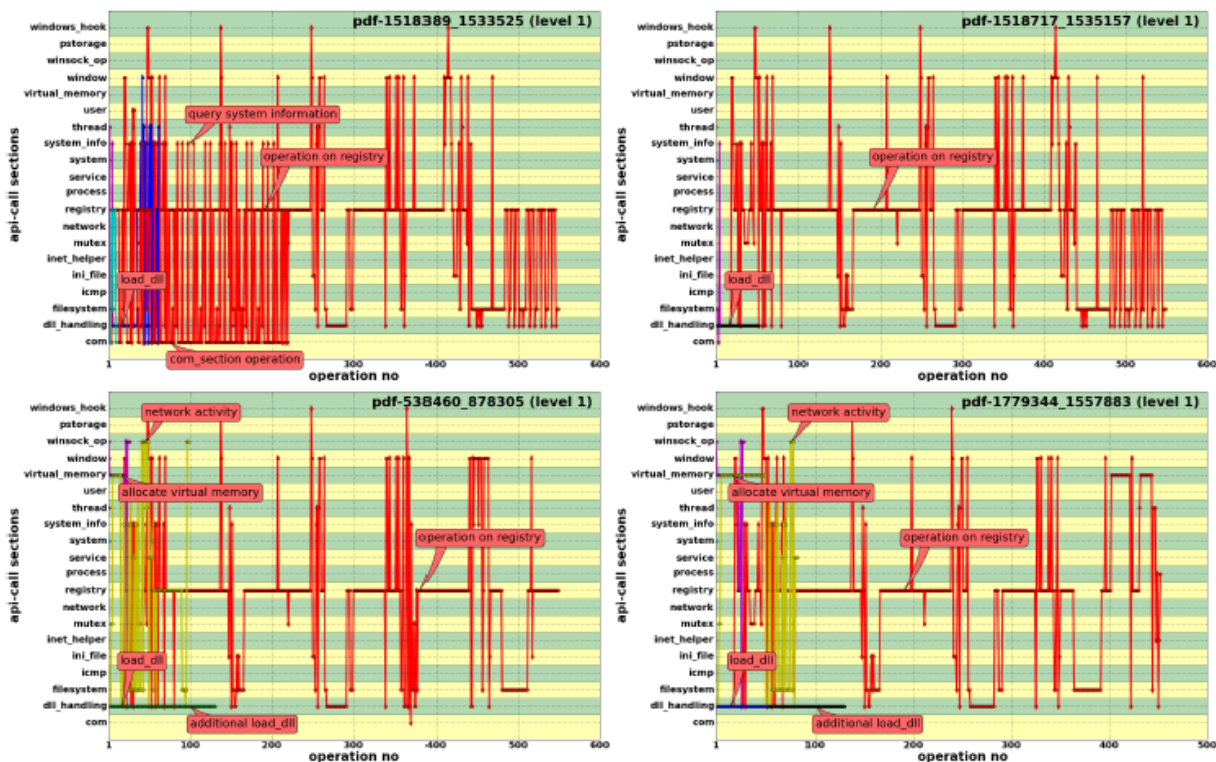


图 9：两个良性 pdf 文件的 level 1 线程图 ( a ) - ( b )、两个恶意 pdf 文件的 Level 1 线程图 ( c ) -(d)

## 7. 结论

本文展示了两种恶意软件行为可视化方法。在对恶意软件进行分析时，我们发现这两种方法均十分有帮助。我们也十分希望其他分析人员能够提出建设性的意见来帮助我们进行相关的改进工作。尽管我们广泛地使用了 CW 沙箱，我们的可视化途径并不依赖于沙箱。

这种行为分析途径的问题是，我们只能看到一种二进制执行路径，可能因此错过分析过程中的重要功能。随着对恶意软件执行情况的多路径化分析，这种限制可以在某种程度上减小。另外，动态分析的缺点也要考虑在内。行为分析的另一个缺点是，被分析的二进制数据可以检测到分析环境，因而表现异常。

我们的研究还可以在某些方面进行扩展，比如，开发某种放大功能，让分析人员在树形图的伸缩之间游刃有余。就像我们在文中展示的全球树形图一样，我们可以将每一个放大系数进行丰富和细化，这样的分析人员就可以对分析报告中不常观测的部分进行详细地探究了。

在未来的工作中，我们还可以对现有的图像聚类 and 分类算法进行研究，将这些算法在我们数据库中的计算结果与反病毒厂商的标签或基于全行为分析报告的聚类进行比较。而且，由于可视化分类比基于详细

报告的分类要快得多，可视化分类可以作为一种预处理手段，甚至对精确的分类进行预测。

## 参考文献

- [1] U. Bayer. Anubis: Analyzing Unknown Binaries. <http://analysis.seclab.tuwien.ac.at/>.
- [2] Gregory Conti, Erik Dean, Matthew Sinda, and Benjamin Sangster. Visual Reverse Engineering of Binary and Data Files. In *Workshop on Visualization for Cyber Security (VizSec)*, 2008.
- [3] Cullen Linn and Saumya Debray. Obfuscation of Executable Code to Improve Resistance to Static Disassembly. In *ACM Conference on Computer and Communications Security (CCS)*, 2003.
- [4] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring Multiple Execution Paths for Malware Analysis. In *IEEE Symposium on Security and Privacy*, 2007.
- [5] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.
- [6] Norman ASA. Norman SandBox. <http://sandbox.norman.no/>.
- [7] Igor V. Popov, Saumya K. Debray, and Gregory R. Andrews. Binary obfuscation using signals. In *USENIX Security Symposium*, 2007.
- [8] Ben Shneiderman. Treemaps for space-constrained visualization of hierarchies. Internet: <http://www.cs.umd.edu/hcil/treemap-history/>.
- [9] Ben Shneiderman. Tree Visualization With Tree-Maps: 2-D Space-Filling Approach. *ACM Trans. Graph.*, 11(1), 1992.
- [10] Symantec. Internet Security Threat Report Volume XIV. Internet: <http://www.symantec.com/business/theme.jsp?thmeid=threatreport>, April 2009.
- [11] Carsten Willems, Thorsten Holz, and Felix Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2), March 2007.
- [12] Ying Xia, Kevin Fairbanks, and Henry Owen. Visual Analysis of Program Flow Data with Data Propagation. In *Workshop on Visualization for Cyber Security (VizSec)*, 2008.