大型恶意软件数据库中
共享系统调用序列关系的可视化

非官方中文译本 • 安天实验室 译注

文档信息			
论文题目	Visualization of Shared System Call Sequence Relationships in Large Malware Corpora		
论文作者	Josh Saxe, David Mentis, Chris Greamo		
发布单位	Invincea Labs		
原文链接/出处	http://www.linkedin.com/profile/view?id=9408813&authType=NAME_SEARCH&authToken=hwOH&locale		
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	**摘要：**我们在大型恶意软件数据库中使用一种自动化识别和交互可视化调用序列关系的共享系统。该系统始于提取可变长度的启发式算法应用，从恶意软件系统调用有意义的语义系统调用序列的行为日志。然后，基于这些语义序列生成，我们构建了一个布尔向量表示的恶意软件样本库。最后在样本向量上计算 Jaccard 系数（相似性系数），从而获得一个样本相似度矩阵。 **关键词：**计算机安全；恶意软件；数据挖掘；数据可视化		
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	• 本译文译者为安天实验室工程师，本文系出自个人兴趣在业余时间所译，本文原文来自互联网的公共方式，译者力图忠于所获得之电子版本进行翻译，但受翻译水平和技术水平所限，不能完全保证译文完全与原文含义一致，同时对所获得原文是否存在臆造、或者是否与其原始版本一致未进行可靠性验证和评价。 • 本译文对应原文所有观点亦不受本译文中任何打字、排版、印刷或翻译错误的影响。译者与安天实验室不对译文及原文中包含或引用的信息的真实性、		

	准确性、可靠性、或完整性提供任何明示或暗示的保证。译者与安天实验室亦对原文和译文的任何内容不承担任何责任。翻译本文的行为不代表译者和安天实验室对原文立场持有任何立场和态度。 • 译者与安天实验室均与原作者与原始发布者没有联系，亦未获得相关的版权授权，鉴于译者及安天实验室出于学习参考之目的翻译本文，而无出版、发售译文等任何商业利益意图，因此亦不对任何可能因此导致的版权问题承担责任。 • 本文为安天内部参考文献，主要用于安天实验室内部进行外语和技术学习使用，亦向中国大陆境内的网络安全领域的研究人士进行有限分享。望尊重译者的劳动和意愿，不得以任何方式修改本译文。译者和安天实验室并未授权任何人士和第三方二次分享本译文，因此第三方对本译文的全部或者部分所做的分享、传播、报道、张贴行为，及所带来的后果与译者和安天实验室无关。本译文亦不得用于任何商业目的，基于上述问题产生的法律责任，译者与安天实验室一律不予承担。

大型恶意软件数据库中
共享系统调用序列关系的可视化

Josh Saxe
Invincea Labs
josh.saxe@invincea.com

David Mentis
Invincea Labs
david.mentis@invincea.co

Chris Greamo
Invincea Labs
chris.greamo@invincea.co

摘要

我们在大型恶意软件数据库中使用一种自动化识别和交互可视化调用序列关系的共享系统。我们的系统流水线开始于提取可变长度的启发式算法应用，从恶意软件系统调用有意义的语义系统调用序列的行为日志。然后，基于这些语义序列生成，我们构建了一个布尔向量表示的恶意软件样本库。最后在样本向量上计算 Jaccard 系数（相似性系数），从而获得一个样本相似度矩阵。

我们的图形用户界面链接内部交互式展示两个可视化效果。第一个视图是基于减少三维投影相似度矩阵。第二个视图提供所选的恶意软件的异同，并深入他们分享的调用序列。我们也提供一套基于恶意行为特征的交互式过滤器。这些视图集成到一个交互、链接的显示器，允许用户理解恶意软件数据库的相似性结构，检查行为特征如何对数据库进行统计分布，并深入检查局部相似性和样本之间的差异。

类别和主题描述符

D.4.6 [安全和保护]：入侵性软件（如：病毒、蠕虫、木马）

一般术语：安全，人工智能，可视化

关键词：

计算机安全；恶意软件；数据挖掘；数据可视化

该工作受 DARPA 网络基因组计划资助公开发行批准，仅表示作者的观点，不代表官方政策、国防部或美国政府的立场。

1. 简介

近几年研究者已应用了恶意软件自动相似性分析技术，帮助解决各种新恶意软件变种在互联网上泛滥情况。尽管在这一研究领域已经诞生了有效的方法，始终需要探索产生更容易解释恶意软件数据库可视化的方法。这里我们提出了一种大型恶意软件数据库中共享系统调用序列关系的可视化方法。

我们的工作基于长期提取语义序列的算法。恶意软件系统调用登录，以便提取的系统调用序列的语义序列分区表，提取有意义的功能块。打个比方，如果系统调用日志被视为长流水句，我们的算法在自然区域标记符号。一旦我们找到这些标记符，我们对日志序列进行标记，视为分成界限的子序列。

我们的序列分区算法基于两个假设： 第一，如果我们是代表 Markov 链的系统日志调用，在特殊情况下增加文件路径、注册表项或操作的网络元组，我们能找到有意义的分区，这些地方不可能发生状态转换。这是因为，我们推测，这种转变将倾向于发生的循环、函数和其他经常重复的控制流结构的末尾。第二，我们本能地知道当在这些日志中发生逻辑中断时，系统调用和其后继大幅偏离文件路径、注册表路径或网络元组相似性。我们对恶意软件数据库，使用我们的分区方法提取可变长度的子序列，表示有意义的程序行为块。接下来，我们依照当前子序列行为日志集，比较样本。

若要展现子序列发生和样本相似之处，我们在原型 GUI（图形用户界面）内链接了大量显示器，GUI 的主面板显示整个恶意软件数据库分析下的相似结构。具体来说，我们采用多功能三维缩放技术，将样本相似度矩阵投射到一个二维网格。

为了支持各种恶意软件分析任务，我们的接口支持大量交互。用户可以在网格高亮显示一个或多样本，在屏幕顶部呈现的检测面板，提出执行选定的恶意软件语义序列的详细视图。左侧的显示面板向用户提供一组过滤器，以便他们可以看到各种行为特征怎样分布在恶意软件数据库中。

本文介绍了我们的系统，以及恶意软件可视化方面的已有研究。本文其余部分的结构如下所示。第二章详细列

出了我们每个系统组件采用的恶意软件沙箱描述、语义序列提取算法、序列标注算法和相似性度量方法。第三章论述了我们系统的可扩展性和在系统开发和测试中使用的测试数据集性质。第四章通过叙述设想我们研究结果的方法，包括讨论样本序列可视化、相似性地图视图和交互式过滤器。第五章叙述系统探索恶意软件相似性用例的数目。对于每个用例，我们基于真实世界的恶意软件举例。第六章讨论了相关工作。第七章讨论了我们未来工作的计划。

2. 系统说明

我们的系统包括四个部分 :提取恶意软件行为日志的虚拟沙箱、语义序列提取和相似度计算组件、将影响系统的语义序列分类的标签组件、向用户显示分析结果的可视化工具。

2.1 插桩环境

我们利用两个现有工具来提取记录恶意软件样本的执行行为：x86 模拟器 QEMU 和 Windows 操作系统插桩工具 Procmon[2] [3]。我们通过两个工具提取了给定恶意软件样本的行为日志，该过程如下所示。

一个样本被上载到模拟 QEMU 沙箱的虚拟硬盘。Procmon 负责收集所有进程的系统调用信息，在虚拟客户机 Windows XP 操作系统上启用。然后，该样本在沙箱中执行。样本在一个全局预定义的时间段运行完成或超时后，从沙箱中提取 Procmon 行为日志，关闭虚拟操作系统。对于本文所述的测试，我们让样本最长运行 10 分钟。

运行样本之后 ,从 Procmon 收集日志 ,我们对 Procmon 日志进行删减，如下所示。首先，我们对系统恶意软件样本分析之下的进程 ID 进行识别。然后我们遍历其系统调用日志 ,确定在何处调用进程或其任何子代中创建新线程和子进程的位置。最后，我们筛选行为日志，以便在日志中只有恶意软件进程及其子代的线程和进程。

我们检查的 Procmon 日志内的系统调用数据包含若干字段，我们在分析中使用其中的 4 个字段。第 1 个字段是线程 ID，我们基于线程将事件排序成列。第 2 个字段是时间戳，该值指示做出的每个系统调用。第 3 个是操作名称，表示系统调用主进程中注册表、文件系统或网络的关系。

第 4 个字段包含关于系统或网络对象操作行为的信息。对于注册表操作，此字段包含完全合法的、用于文件操作的注册表键值路径，以及用于网络操作、相关源 IP 地址、源端口、目标 IP 地址和目标端口的完全合法的文件路径。Procmon 只记录 TCP 和 UDP 的通信，而不记录 ICMP 通信流。因此 ,目前我们分析只考虑了 TCP 和 UDP 的网络通信。应该注意的是，在本文中，系统调用是指 Procmon “操作” 字段；Procmon 采用由 Windows 操作系统提供的实际系统调用的略微抽象的版本来填充此字段。

2.2 语义序列提取

一旦我们从恶意软件样本数据库中收集了行为日志，我们就采用新技术 ,将每个系统调用日志样本分成系统调用序列功能块（比如遍历注册表键值树以查找键值入口，或通过 ReadFile 和 WriteFile 将一个文件从一个位置复制到另一个位置的循环）。

如上文所述 ,我们的算法基于两个独立的假设。首先，我们假设如果从恶意软件样本中系统调用日志源自 Markov 链，我们能够在系统调用间不可能进行状态转换的点有意义地划分日志。我们认为，这是因为不可能的状态转换表示程序循环终止和进出函数(从恶意源代码的多个地方调用) 的终止。我们第二个假设是，当系统调用和其后续系统调用涉及高度分化的文件、网络对象或注册表项的时候，有意义的分区可能存在。

我们的序列提取算法涉及以下步骤。在初始的预处理步骤中，我们将恶意软件行为日志通过线程 ID，然后按照时间戳，以 “平整” 日志的线程结构。接下来，我们从行为序列结果集导出 Markov 链。此链中的节点被定义为独特的系统调用及其作用于的系统对象，例如 ，作用于 “c:\temp.txt” 的 “WriteFile” 将构成节点。转换率通过将我们观察到的给定节点和其后续节点之间的转换次数相加而获得。为了获得边的转换率，我们用转换总数除以边及其所有相邻边的总数。

基于恶意软件行为构建 Markov 链后，我们可以对日志进行另一种处理 ,计算每个系统调用二元的其他参数相似性分值。在高级别上，这个分值通过比较每个系统调用及其随后系统调用的参数字符串来计算；它们越不同，则分值就会越低。具体而言，两个连续调用之间的相似性计算方法如下。如果这两次调都在注册表中，我们用注册表键值参数分隔符 “\” 来标记其注册表路径。然后，我们将得到的注册表路径令牌小写标准化 ;计算两个令牌集合之间的 Jaccard 系数，我们把这个看作参数字符串之间的相似度。

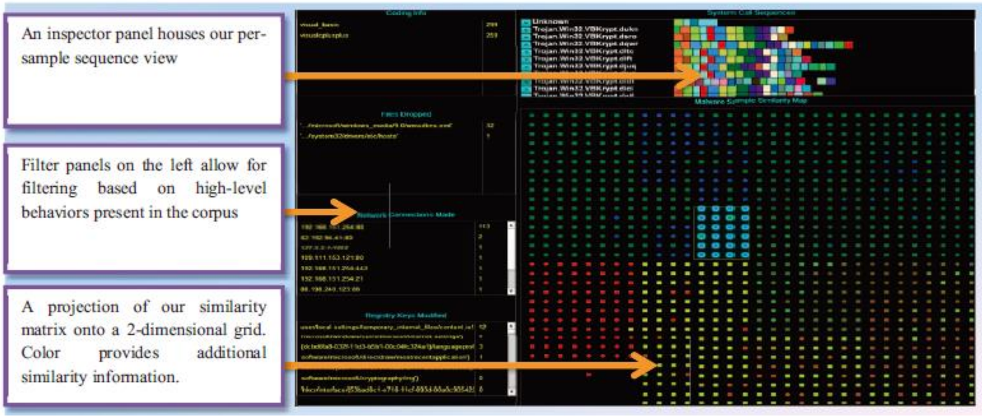

图 1：我们的原型 GUI

Example Semantic Subsequence Extraction from Sample of Variant Type "menti-gtmr"

...

ReadFile C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll
ReadFile C:/WINDOWS/WinSxS/x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.5512_x-ww_dfb54e0c/GdiPlus.dll

---

RegQueryKey HKLM/SOFTWARE/Microsoft/Windows-NT/CurrentVersion/Fonts

---

RegCreateKey HKU/S-1-5-21-436374069-813497703-1177238915-1004/Software/Microsoft/GDIPlus

---

RegQueryValue HKU/S-1-5-21-436374069-813497703-1177238915-1004/Software/Microsoft/GDIPlus/FontCachePath

---

QueryOpen C:/RUNME/ShFolder.DLL
QueryOpen C:/WINDOWS/system32/shfolder.dll
CreateFile C:/WINDOWS/system32/shfolder.dll
CreateFileMap C:/WINDOWS/system32/shfolder.dll
CreateFileMap C:/WINDOWS/system32/shfolder.dll
Load Image C:/WINDOWS/system32/shfolder.dll

...

图 2：我们的样本序列提取算法的示例输出，水平线表示分区。

如果文件执行了两次调用，则我们对文件路径执行相同的令牌标记操作。如果调用涉及网络操作，如果涉及相同的外国 IP 地址和端口，则我们把它们的相似度视为"1"；如果涉及不同外国的 IP 地址和端口，则相似度为"0"。最后，如果是两次不同类别的调用（例如，注册表调用和文件调用），则我们将它们的相似度设置为 0。我们这样做是因为，至少现在我们还没有更复杂的、比较这两种不同参数类型的方法。

计算转换概率和系统调用日志中每个系统调用转换的参数相似度后，我们最后一次遍历该日志，并插入分区和提取语义序列。要决定是否在任何特定的系统调用中插入分区，我们计算系统调用转换参数相似度的分值和其转换率之间的加权平均数，然后确定该平均数是否低于阈值，在本文中，我们将阈值设为 0.3。

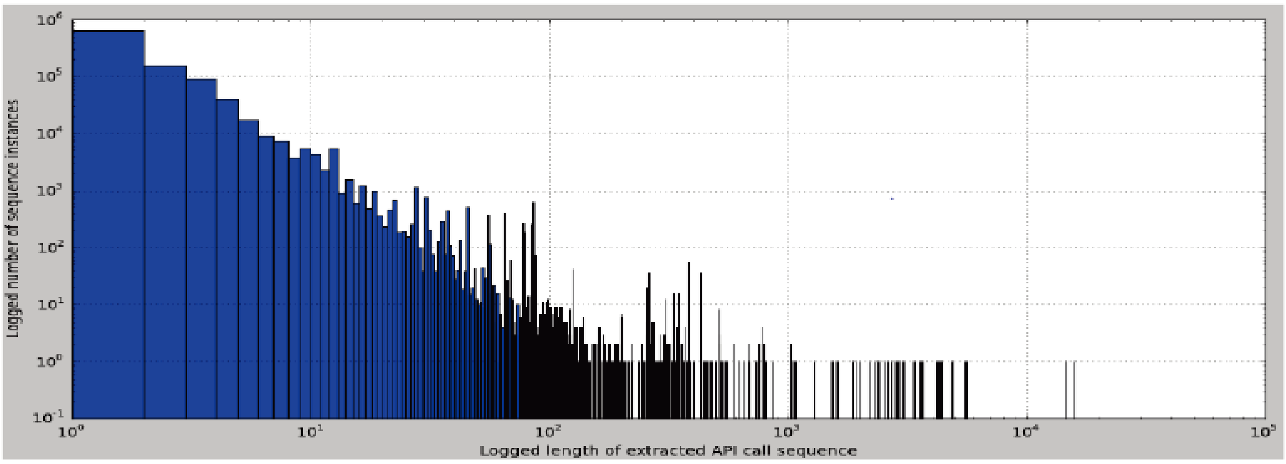

图 3：从恶意软件的可执行文件中抽样提取出来的语义序列长度的直方图

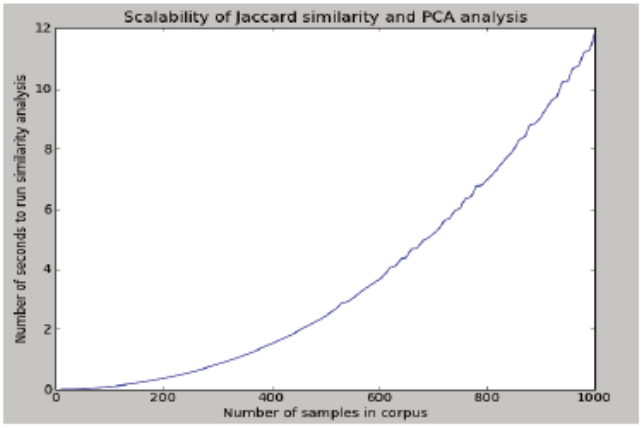

图 4：单个 Intel Xeon e5645 内核（32GB RAM,2.4GHz）运行时数据库大小的影响

如果分值低于阈值，我们认为系统调用日志中该位置的程序行为存在功能阻碍，我们会插入一个分区。当我们插入分区后，我们提取分区之间的系统调用部分，将其作为语义序列。在本文讨论的试验中，我们参数相似度分值和转换概率对最终的加权平均数(我们根据该平均数确定是否分区)是同等重要的。在未来，我们打算研究加权平均数和阈值参数的最优组合。

图 3 显示了我们从 1000 个恶意软件样本的数据库中提取的语义序列长度的直方图，我们将在第三章中介绍。图 3 显示这些序列往往是趋向短小，并且遵循逆幂律长度分布。

2.3 序列标签

为了支持可视化，我们使用易于阅读的文本标签来标记从恶意软件数据库中提取的序列。这些标签都基于简单的正则表达式(与系统调用参数匹配)分配。例如，如果一个序列中的系统调用用子字符串“registry”和“Internet”修改了注册表键值，则我们创建一个标签，将该序列描述为修改 Internet 设置。一个序列可以有一个或多个标签，用来向 GUI 中添加描述性信息。我们计划在今后详细研究系统调用的语义序列分类。

2.4 相似度矩阵计算

从恶意软件数据库中提取变长序列后，我们会进行相似度矩阵计算。具体来说，我们基于每个样本中变长序列的出现来构建布尔向量样本。为了消除噪音，我们只考虑至少在两个样本中出现的序列。最后，我们对样本向量成对地计算 Jaccard 系数。正如我们如下讨论地，提取的子序列和相似性样本随后用于交互可视化。

3. 数据库和性能

我们一直在测试系统对恶意软件数据的防御。具体来说，我们在系统上测试从 MD:Pro 恶意软件库中收集的样本，并检查样本是否对应卡巴斯基的样本反病毒引擎分类，从而验证我们的算法。下面，我们将介绍该算法的数据库、标准和运行时间；以及与标准相比时我们系统的精度。

3.1 数据收集

本文描述的所有恶意软件样本都来自 MD:Pro。MD:Pro 是由 Frame4 安全咨询公司提供的付费订阅恶意软件服务。2011 年 2 月至 2012 年 6 月，我们从 MD:Pro 收集了 100,415 个独特恶意软件样本，到目前为止，我们在恶意软件分析沙箱中运行了 28,460 个样本。此外，我们将每个样本输入到卡巴斯基反病毒引擎来生成标准标签。卡巴斯基将这 30,104 个样本按照强大的分类标准(如“trojan.win32.jorik.skor.akr”)进行分类，认为 41,830 个样本属于“未知”，而其他 28,481 个样本则被分配了低保真度的标签 (如“generic win32 trojan”)。

我们从订阅的 MD:Pro 恶意软件中精心挑选了一个数据库，用于本文的测试和示例。由于卡巴斯基标签的不一致(一些卡巴斯基标签分组，如“jorik”家族，表示宽泛的恶意软件；而其他家族，如“webprefix”，则表示非常特定的变种)，所以我们启发式地选择了我们认为共享类似属性的 4 个卡巴斯基定义的恶意软件类别。具体来说，我们选择了“xtoober”、“jorik”和“vbkrypt”家族，并随机选择了一组被卡巴斯基列为“未知”的恶意软件样本。

3.2 验证

为了验证我们的序列特征和 Jaccard 相似性函数能够有效地捕获恶意软件的相似性，我们已经验证了它们反映了恶意软件分析人员确定的标准。即，我们筛选了仅由卡巴斯基反病毒引擎分类的样本数据库，计算这些样本的相似性矩阵，聚类样本相似度矩阵，计算精度，然后统计我们的聚类与卡巴斯基分类的匹配度。

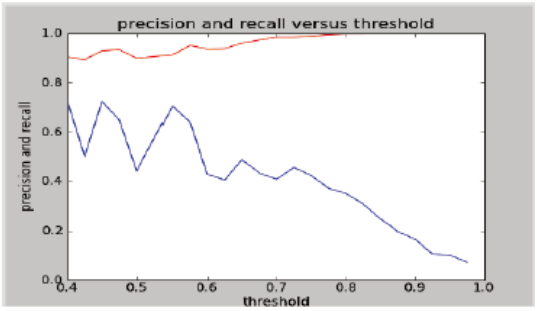

图 5：聚类性能随着相似性阈值增加而变化，红线表示精度和蓝线表示回调。

我们通过加权平均数来计算每个聚类的精度和回调率，其中权重是每个聚类所包含的样本的数量。更多关于

精度和回调率的措施，请参考文献[4]。

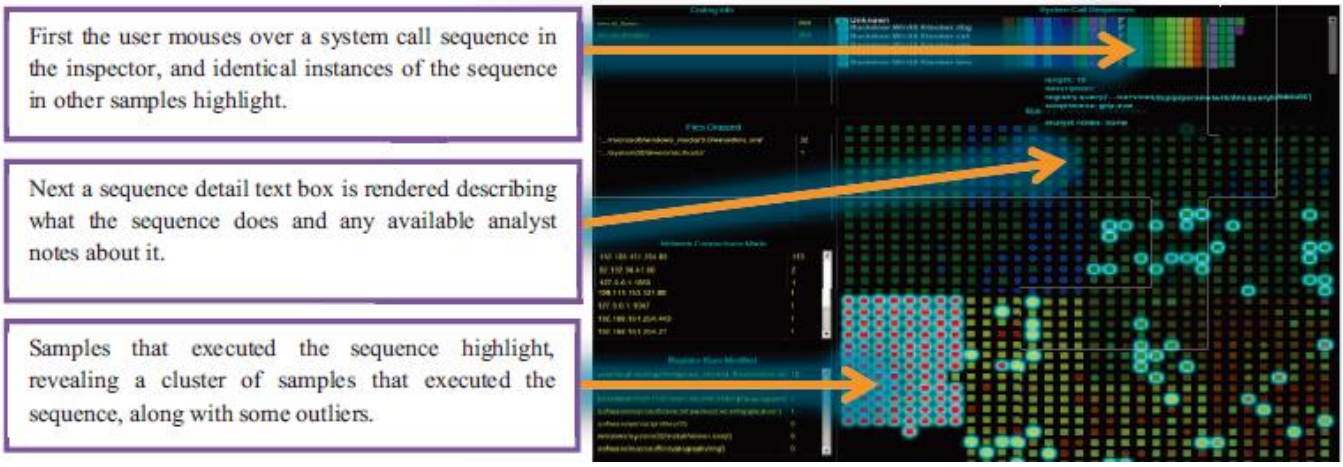

图 6：序列和相似性视图之间语义联系的例子

我们使用图论方法来聚类恶意软件样本向量。该方法如下所示。首先，我们定义一些相似性阈值 t ,这样通过阈值反映边的相似之处，我们构造相似度矩阵图。然后我们可以确定图中的最大子集，将其从图中删除，重复这一过程，直到整个图完全分解为最大子聚类。这种算法的意图是：最大的聚类共享相似度大于阈值 t 的所有边。因此每个节点至少保证与聚类中的其他节点共享 t 百分比的序列。

图 5 显示了增加聚类算法相似性阈值时精度和回调分值的变化。当相似性阈值是 0.8 时，我们可以与卡巴斯基恶意软件分类完美地匹配样本；但对于许多聚类，我们未能检索到相关类别。另一方面，当阈值为 0.55 时，我们可以检索每个聚类中的样本 ,我们能够将大部分样本正确分类。这些结果表明我们的语义序列特征可以有效地高精度地区分多个恶意软件类别。

3.3 算法运行时间

由于恶意软件规模增长，我们的序列提取算法（基于每个样本）也呈线性扩展。我们发现，在包含 917 个恶意软件样本系统调用痕迹的测试中，该算法平均花费 0.34 秒来提取每个样本的序列。我们的数据库样本平均有 311.67 次系统调用。

图 4 说明了我们的相似度矩阵和主要组件分析步骤如何随着数据库增长而增加。当我们的算法规模以指数方式扩展 ,我们可以在几个小时内处理数千或数万的恶意软件样本。未来，我们计划扩展该系统，以处理数万甚至数十万的样本。

4. 可视化

我们的接口包含 3 个面板：1 个序列检查器，该检查器检查在恶意软件执行期间发生的实际子序列；1 个代码共享映射图，提供恶意软件样本行为相似性的全局视图；1 组过滤器，允许用户查看恶意软件数据库的特征关系。这 3 个显示器相连，因此，例如，只要在序列检查器中输入，序列的其他出现也会被高亮显示，代码共享映射图中的所有样本也是如此（参见图 6）。

4.1 样本相似度映射

我们的相似度可视化示图以二维网格的形式呈现。我们在样本相似度矩阵中执行主要组件分析 ,从而计算样本相似性网格的布局和着色。我们将前 3 个主要组件投射到一个减少的维度（三维）空间。然后我们根据每个样本第一个主要组件的值来分类样本，并将其排列于空间填充 Hilbert 曲线。我们计算每个节点的位置，然后给 3 个主要组件着色，来确定每个颜色的红、绿和蓝色值。类似样本紧邻，并显示相似的颜色。目前，节点可以是圆形或正方形，圆圈代表“已知”样本（我们相应的标准），而方块代表未知的样本（卡巴斯基反病毒引擎还未能标记）。

我们选择使用一个 2d 网格布局 ,并使用颜色通道来编码其他相似性信息，来处理更传统、散点图多维度缩放技术导致的两个问题。首先，网格布局避免重叠节点的问题。我们认为如果用户不必直观地解析重叠节点，则可视化会更有效。第二，网格布局最大化了屏幕上的可用像素的使用。在未来的研究中，每个节点的这些信息将允许我们使用节点形状来编码其他节点相似数据。

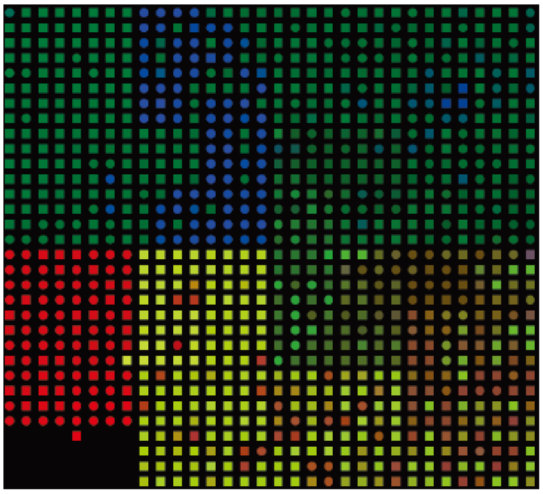

图 7：相似性网格布局中的恶意软件样本，颜色表示空间维度，大量聚类可以通过颜色形状识别。

4.2 序列可视化

序列可视化揭示了恶意软件样本的异同。如图 8-11 所示，我们将样本数据库的子序列呈现如下。我们为每个独特序列分配唯一颜色。然后我们通过全局唯一标识符（在本例中，是指该序列的数据库 ID），将其呈现为顺序的颜色块，其宽度随着长度的自然长度缩放。每个序列的对数缩放使我们能够更有效地使用显示屏。

图 6 显示了我们与可视化表示的交互方法：如果一个恶意软件的序列被输入过，则相同和其他样本中相同的序列也会被高亮显示，同时也会生成详细的信息，提供有关该序列的说明信息。

我们选择使序列可视化作为每个命令所支持的唯一标识符的示例，为了支持比较一组出现在每个样本的执行痕迹的序列可视化。图 8 阐释了这种比较方式的有序排列。遗憾的是，以这种方式呈现的数据，并不显示序列中那些是重复的，案例中重复序列的具体情况取决于出现的样本。我们希望在今后的工作中，探讨如何解决这一问题。

4.3 过滤器显示屏

左边显示提取恶意软件样本库的行为特征的名称。目前，我们支持的特征类型（如注册表键修改，网络通信和丢失的文件。恶意软件样本表现出掠过的动作特征类型突出显示。这种连接过滤器和代码共享地图之间，支持用户在不同区域图上深入了解比较区域间的异同。图 12 显示了此功能。

5. 案例

5.1 可视化地识别和研究恶意软件中的聚类结构

相似性映射可视化揭示了恶意软件样本库中的聚类机构。在图 7 中，可以清楚看到大量的恶意软件聚类，例如：红、黄、绿和蓝组。另一方面，映射图的右下象限出现了模糊和扩散，表明它缺乏恶意软件样本之间明确的聚类结构。因为这张映射图是一个多高维度的空间到几个维度上的投影，不可避免地会有一些信息丢失。这样就允许用户突出显示样本并检查其序列集合是充分利用映射图的关键。

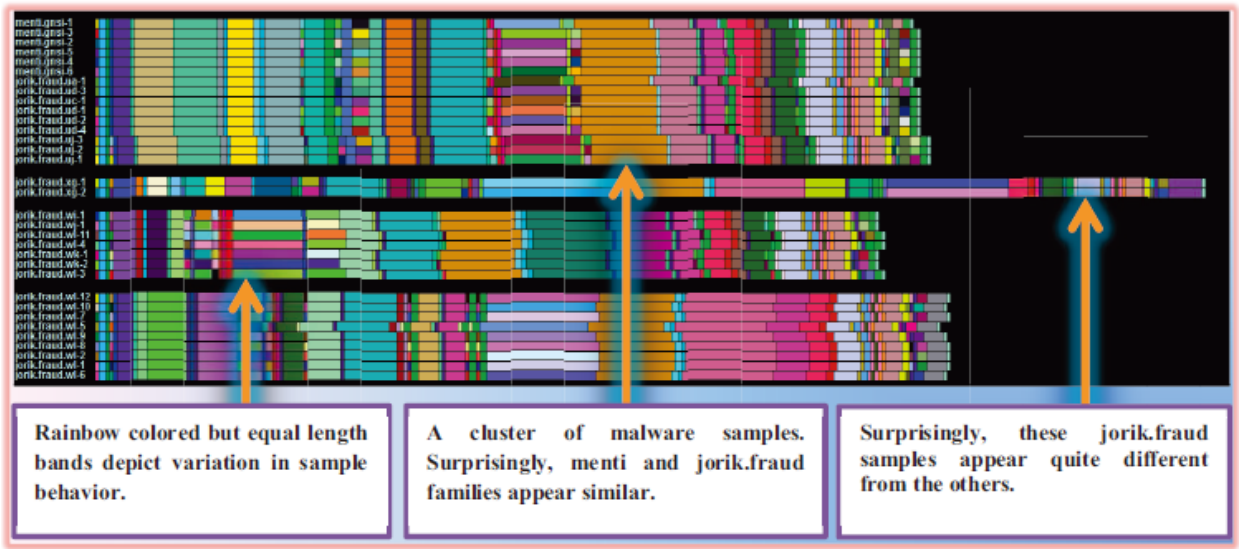

图 8：用户在左边输入 IP 地址和端口，与 IP 地址和端口相关的样本在右边。

5.2 探索数据库的行为特征关系

过滤器左面板的界面允许用户发现如何分布在恶意软件数据库的各种行为特征。图 12 中，用户输入 IP 地址和 TCP 端口，与其相关的样本就会显示；很明显恶意软件在左下角的相似性映射聚类的特点是它与该地址和端口相关。能够进行筛选，发现聚类和特征之间的关系，有助于分析并洞察相似度映射图的意义。

5.3 将未知和已知恶意软件样本联系起来

我们发现系统在涉及新型样本和已知的恶意软件样本方面很有效。如图 9 所示，未知和已知恶意软件样本的相似度映射图紧密相关（左边，标签显示）。我们的系统

揭示了未知的样本（卡巴斯基反病毒引擎无法分类）行为非常类似于 xtoober 样本。要在序列检查器中获得更多关于样本的信息，用户可以鼠标悬停在序列检查器屏幕顶部（如图 6 所示）中显示序列。

5.4 探索恶意软件样本之间的序列共享关系

该系统使得用户能够可视化地深入观察执行系统调用序列的样本和样本聚类。图 6 显示了用户从样本相似度映射图中选择了一组样本，然后在序列检查器中输入一个序列。序列检查器中的该序列的所有的出现都会高亮显示，表明该序列由所有目前正在检测的样本共享。此外，所有数据库中的高亮样本映射在相似度映射图中。在这种情况下，焦点序列似乎主要发生在密集排列的聚类和高相似度样本。

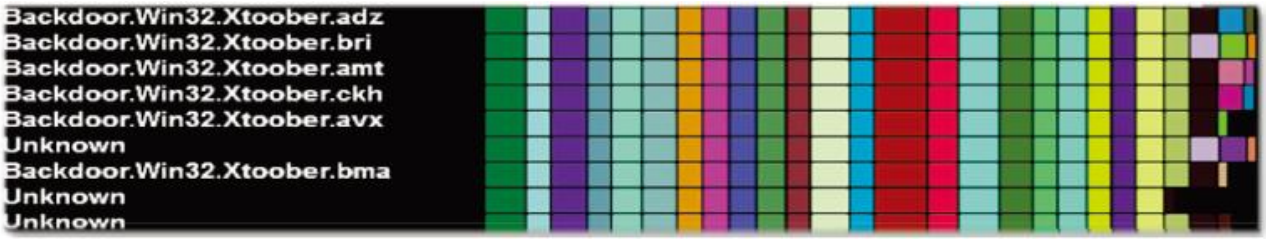

图 9：已知与未知恶意软件样本的可视化聚类，未知样本的行为序列几乎与标记样本相同。

图 10：比较未知和已知样本，两个样本之间显然存在重叠。

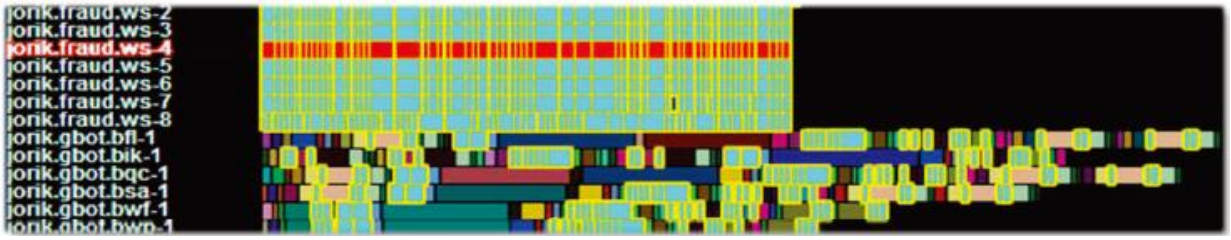

图 11：其他样本有相似之处 jorik.fraud.ws 4 的序列会高亮显示。

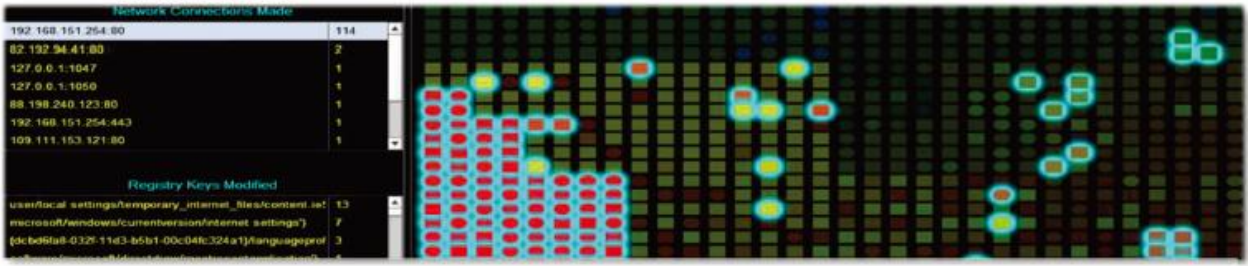

图 12：用户在左边输入 IP 地址和端口，右边高亮显示与该 IP 地址和端口相关的样本。

5.5 支持分析继承和重用

当用户输入序列时，会出现描述该序列中至少一个或几个系统调用的标签的提示。这也是为分析人员提供的注释。目前这些并未使用，但在今后的工作中，能够支持分析序列注释，这样，当新样本显示包含这些分析序列的新样本时，可以看到这些序列已经被“分析”了，并且能够受益于以前的分析工作。

6. 相关研究

据我们所知的，我们的成果首次探讨了大型恶意软件数据库的交互式可视化研究。在机器学习领域，Storlie 等人探讨了使用 Markovian 随机邻接矩阵执行跟踪恶意软件动态[8]。我们的工作侧重与 Markovian 方法区分的系统调用序列。

Conti 等人介绍了支持分析理解的二进制文件结构的可视化技术的相关可视化成果[5]，Conti 的工作期望于展示比较我们关注低抽象系统调用序列。Trinius 等人探讨了应用 treemaps 和线图进行比较分析恶意软件[6]。而他们专注于恶意软件执行系统调用类别的相对数量动态分析，我们专注于共享系统的调用序列关系。而他们专注于对小型恶意软件样本集合之间的相似度的探索提供支持，我们专注于一个提供大型恶意软件样本的接口，用户可以深入研究一组选定样本，并比较它们之间的相似度。

Quist 等人创造了一个提供恶意软件执行跟踪的交互式可视化和在 CPU 指令级别执行跟踪元数据的系统[7]。而这项成果的重点是对支持个别样本在汇编语言级别逆向工程的工作，我们的成果支持对跨系统调用级别样本的逆向工程。

7. 未来的工作

为了扩展我们所做的工作，我们打算评估若干方法并有尽可能改善它们的情况。我们目前使用 Hilbert 曲线展现我们相似度网格的示例，但这可能不是对于我们数据的最佳网格布局，并且我们计划对多维度和网格布局上探讨了现有的文献。我们也想探索将我们的系统纳入多种相似性度量，这样用户能够，例如，切换序列相似度、调用图形编辑距离相似度和可输出的字符串相似度。最后，我们计划采取从原型到更加强大并且有能力处理系统扩展用例的一系列状态。

8. 致谢

我们希望对国防部先进研究项目局（DARPA）表示感谢，DARPA 向我们提供了资金，使这项工作成为可能。

9. 参考文献

[1] A.-T. Institute, "AV-Test Statistics Report," 2012. [Online].

Available: http://www.av-test.org/en/statistics/malware/.

[2] "QEMU," 1 June 2012. [Online]. Available:

http://wiki.qemu.org/Main_Page. [Accessed 1 June 2012].

[3] M. Russinovich and B. Cogswell, "Microsoft TechNet,"

Microsoft, 1 6 2012. [Online]. Available:

http://technet.microsoft.com/enus/

sysinternals/bb896645.aspx. [Accessed 1 6 2012].

[4] P. R. H. S. Christopher D. Manning, 2008. Introduction to

Information Retrieval, Cambridge University Press, 2008.

[5] G. Conti, E. Dean, M. Sinda, B. Sangster and J. Goodall,

2008. Visual Reverse Engineering of Binary and Data Files.

Visualization for Computer Security, Springer Berlin /

Heidelberg, 2008, pp. 1-17.

[6] P. H. Trinius and J. a. F. T. Gobel, 2009. Visual analysis of

malware behavior using treemaps and thread graphs. In

Proceedings of the International Workshop on Visualization

for Cyber Security (VizSec 2009).

[7] D. Quist, 2009. Visualizing compiled executables for

malware analysis. In Proceedings of the International

Workshop on Visualization for Cyber Security (VizSec 2009).

C.

Storlie, S. V. Weil, D. Quist, B. Anderson, 2012.

Stochastic Identification and Clustering of Malware with

Dynamic Traces. Malware Technical Exchange Meeting

2012.