

使用可视化图像矩阵进行恶意软件分析

非官方中文译本 · 安天实验室 译注

| 文档信息 | | | |
|------------|---|--------|-----------|
| 论文题目 | Malware Analysis Using Visualized Image Matrices | | |
| 论文作者 | KyoungSoo Han , BooJoong Kang , Eul Gyu Im | | |
| 发布单位 | 《世界科学杂志》 | | |
| 原文链接 / 出处 | http://dx.doi.org/10.1155/2014/132713 | | |
| 论文发布日期 | 2014/7/16 | 译文发布日期 | 2014/9/30 |
| 论文摘要 & 关键词 | <p>摘要：本文提出了一种新型的恶意软件可视化分析方法，该方法不仅包含将二进制文件转换成图像的可视化方法，还包含这些图像之间的相似度计算方法。该方法利用从恶意软件样本中提取的操作码序列生成图像矩阵的 RGB 颜色像素，并计算图像矩阵的相似度。</p> <p>关键词：恶意软件；可视化；图像；矩阵；相似度</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。 | | |

研究文章

使用可视化图像矩阵进行恶意软件分析

KyoungSoo Han,¹ BooJoong Kang,² and Eul Gyu Im³

¹ 韩国, 首尔, 汉阳大学, 计算机与软件系, 133-791

² 韩国, 首尔, 汉阳大学, 电子与计算机工程系, 133-791

³ 韩国, 首尔, 汉阳大学, 计算机科学与工程系, 133-791

请与作者 Eul Gyu Im 通信; imeg@hanyang.ac.kr

2014 年 3 月 14 日收到; 2014 年 5 月 19 日接受; 2014 年 7 月 16 日发布

学术责任编辑: Fei Yu

版权所有©2014, KyoungSoo Han 等人。这是知识共享署名许可下的可访问文章, 在原文适当引用的前提下, 允许在任何媒介下无限制地使用、传播和复制本文。

本文提出了一种新型的恶意软件可视化分析方法, 该方法不仅包含将二进制文件转换成图像的可视化方法, 还包含这些图像之间的相似度计算方法。该方法利用从恶意软件样本中提取的操作码序列生成图像矩阵的 RGB 颜色像素, 并计算图像矩阵的相似度。特别是, 该方法可用于压缩的恶意软件样本, 即通过动态分析将其运用到提取的执行痕迹。当生成图像后, 我们可以只从包含诸如功能和应用程序编程接口 (API) 调用相关的指令的块中提取操作码序列, 从而降低负载。此外, 我们提出了一种能够为每个恶意软件家族生成代表性图像的技术, 以减少未知样本类别和图像矩阵的颜色像素信息 (用于计算图像相似度) 之间的比较次数。我们的实验结果表明, 恶意软件的图像矩阵可以有效地静态和动态地分类恶意软件家族, 其准确率分别为 0.9896 和 0.9732。

1. 简介

恶意软件作者已通过各种手段生成了新的恶意软件及恶意软件变种, 如重用模块或使用自动化的恶意软件生成工具。某些恶意行为模块被重复用于变种中, 所以同一个家族的恶意软件变种可能具有类似的二进制模式, 这些模式可以用于检测恶意软件并对恶意软件家族进行分类。此外, 大多数反病毒程序主要采用恶意软件特征 (也就是字符串模式) 来检测恶意软件 [1]。然而, 诸如混淆或压缩技术的各种检测规避技术被应用到恶意软件变种中, 以规避基于特征的反病毒程序的检测, 使得安全分析人员难以检测到恶意软件 [2, 3]。在恶意软件生成技术的帮助下, 恶意软件的数量逐年增加。

尽管安全分析人员和研究人员一直在研究各种分

析技术来处理恶意软件变种, 但是无法完全分析它们, 这是因为采取规避技术的恶意软件正在呈指数形式增加。因此, 新的恶意软件分析技术应该减少安全分析人员的负担。最近, 一些恶意软件可视化技术已经被提出, 以帮助安全分析人员分析恶意软件。

在本文中, 我们提出了一种新的方法来可视化地分析恶意软件, 并将恶意软件家族分类。该方法将提取自恶意软件的操作码序列转化为图像 (称为图像矩阵), 并且计算图像之间的相似度。此外, 该方法适用于通过动态分析提取的执行痕迹, 所以, 我们可以用来分析采用诸如混淆和压缩等规避技术的恶意软件。为了减少计算负载, 我们使用主要块选择技术, 只从与关键行为有关的块提取操作码序列, 例如功能和应用程序编程接口 (API) 调用 [2]。每个恶意软件家族的代表图像都会生成, 并用于未知样本的迅速分类。使

用这些图像矩阵,我们量化图像的 RGB 颜色像素信息并计算像素的相似度,从而获得图像之间的相似度。

本文的组成部分如下。在第 2 章中,我们介绍了恶意软件分析相关的研究。在第 3 章,我们提出了利用可视化操作码序列的恶意软件分析方法和计算相似度的方法,实验结果在第 4 章呈现。最后,第 5 章给出了结论和未来的发展方向。

2. 相关研究

在一般情况下,用来检测和分类恶意软件的分析方法可以被归类为静态或动态分析[4]。在恶意软件的静态分析中,各种方法,如控制流图(CFG)分析[5-7]、调用图分析[8,9]、字节级分析[10]、基于指令的分析[11-14]以及基于相似度的分析[15, 16]已经被提出了。

通过将提取的指令反汇编成块,并连接块之间的有向边,从而生成 CFG。使用这些 CFG 作为特征的一些恶意软件分析方法已经被提出了。Cesare 和 Xiang[5]提出了一个方法,将 CFG 定义为字符串形式的特征,包含有序节点的图形边的清单,并使用 Dice 系数算法[17]测量特征之间的相似度。Bonfante 等人[6]提出了通过语法和语义分析将 CFG 转换成基于树形的有限状态机的方法,然后将其作为特征。Briones 和 Gomez [7]提出了一种基于 CFG 的自动分类系统。CFG 被归纳为 3 个数组,包括基本块的数量、边的数量、子调用的次数,然后将两个函数进行对比。然而,如果复杂的信息被归纳为很小的量,可能会出现高误报。

大量的研究旨在根据各种信息(例如系统调用、函数、API 调用)检测恶意软件,这些信息在操作系统中用于恶意软件的执行。Shang 等人[8]提出了生成函数调用图的方法,代表了函数调用者和被调用者之间的关系,并将此作为恶意软件样本的特征,然后他们利用这些函数调用图特征计算相似度。Kinable 和 Kostakis[9]通过使用图聚类技术将恶意软件分类。他们提出的方法生成针对恶意软件样本中包含的函数的调用图,并且通过图形编辑距离算法计算出调用图的结构相似度分值,并在此基础上进行聚类。

通过反汇编提取出的指令的统计信息可以在恶意软件的静态分析中使用。Rad 和 Masrom[11]提出了基于指令频率来分类变形恶意软件的方法。由于指令频

率大多是不改变的(即使恶意软件采用了混淆技术),所以指令频率可以作为恶意软件的特征。因此,他们提出的方法使用从恶意软件样本提取的指令频率来计算距离,然后根据距离值划分变形恶意软件。Bilar[12]指出,不同的恶意软件有不同的指令频率。特别是,他们发现,恶意软件中的罕见指令可以成为更好的恶意软件分类预测指标。Han 等人[13]提出了利用指令频率的方法。该方法生成的指令序列根据指令频率进行分类,他们指出,相同的恶意软件家族的指令序列之间的距离具有低距离值。Santos 等人[14]提出了使用 n-gram 指令频率的恶意软件分类方法,其中 n-gram 指令包括 n 指令。在所提出的方法中,他们为每个 n 指令序列生成了向量,并将该向量作为特征。

此外,动态分析方法已经被提出,包括污点、基于行为的方法和 API 调用监控。Egele 等人[18]提出了使用污点技术的方法,跟踪任何浏览器助手对象(BHO)处理的信息流的行为。如果 BHO 泄漏敏感信息,BHO 就会被归类为恶意软件。Fredrikson 等人[19]提出了一种方法,使用图挖掘技术自动地提取行为特性。该方法识别同一恶意软件家族中每个类似的恶意行为的核心 CFG,从而进行聚类,然后将这些概括为重要特征。此外,使用仿真器进行动态监控的方法已经被提出。Vinod 等人[20]通过仿真器中的动态监控跟踪恶意软件的 API 调用,并测量其频率,提取关键的 API。Miao 等人[21]开发了一种被称为“API 捕获”的工具,能够自动提取主要特征,如系统调用的参数、返回值和错误状况。

尽管有许多静态和动态分析方法可用,但仍然需要新的技术来补充现有的技术,以提高安全人员分析恶意软件的能力和便利性。最近,一些可视化的方法被提出,以帮助安全分析人员观察恶意软件的功能和行为[22]。Trinius 等人[23]提出了可视化 API 调用的百分比的方法,并将恶意软件的行为可视化两个图像,分别称为“树形图”和“线程图”。Saxe 等人[24]开发了能够生成两个图像的系统。一个图像显示从恶意软件系统调用行为日志中提取的系统调用序列,而另一个图像显示所选样本之间的相似度和差异。Conti 等人[25]提出了一种可视化系统,能够显示恶意软件样本字节信息的图像,例如字节的值、字节存在、复制的字节序列等。Anderson 等人[26]提出了一种显示恶意软

件样本之间的相似度的方法,该方法采用名为“热图”的图像。Nataraj 等人[27]将字节信息转换为灰阶图像,并使用图像处理划分恶意软件。使用字节值生成图像之后,它们对图像采用抽象表示技术,也就是说,GIST[28, 29],从而计算结构特征。此外,他们证明了利用图像处理的二进制结构分析技术能够比现有的恶意软件分类方法更快速地分类恶意软件[30]。然而,

由于结构分析方法具有较大的计算负载,该方法面临着处理大量恶意软件的问题[31]。

在本文中,我们提出了使用图像矩阵来可视化地表示恶意软件的新颖分析方法,这样能够容易地检测恶意软件的功能,而且可以比其他可视化方法更加迅速地计算不同恶意软件样本之间的相似度。

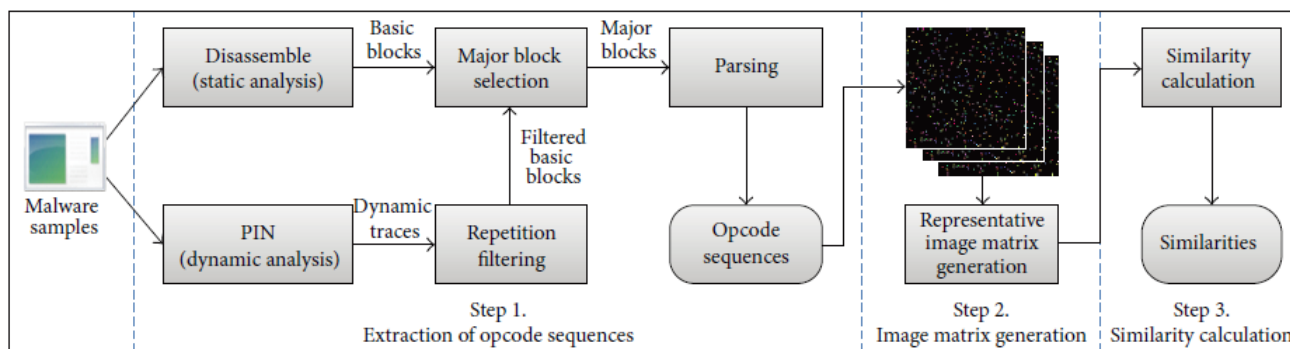


图 1: 该方法的概览

3. 我们提出的方法

3.1 概述。我们提出的可视化恶意软件分析方法由 3 个步骤组成,如图 1 所示,在步骤 1 中,从恶意软件二进制样本或动态执行痕迹中提取操作码序列。步骤 2 生成图像矩阵,其中操作码序列被记录为 RGB 颜色像素。步骤 3 计算图像矩阵之间的相似度。在以下各节中,我们将对每个步骤进行详细说明。

3.2 提取操作码序列。图 2 显示了通过静态分析和动态分析从恶意软件二进制样本或动态执行痕迹中提取操作码序列的过程。

3.2.1 基本块提取。要从恶意软件二进制样本提取操作码序列,需要首先对二进制样本文件进行反汇编,使用反汇编工具(如 IDA Pro[32]或 OllyDbg[33])将其反汇编为基本块。但是,如果恶意软件样本使用了混淆或压缩技术,则使用反汇编器进行静态分析是不可行的[34, 35]。因此,一些恶意软件样本(使用混淆或压缩技术)需要在动态分析环境[36]中执行。

在动态分析中,如图 3 所示,一些重复的指令序列都包括在动态执行痕迹中,因为程序可以进行循环或重复调用,并且这些重复序列可以增加执行痕迹和处理负载的大小。Kang 等人[37]提出了一种动态执行痕迹的重复过滤方法。在采用重复过滤方法之后,我们从动态执行痕迹中提取过滤的基本块。

最后,如果基本块是从病毒样本或动态执行痕迹中提取的,则采用我们的方法从基本块中提取主要块。我们在下一节中介绍。

3.2.2 主要块选择。在本文提出的恶意软件分析方法中,我们从二进制反汇编结果或动态执行痕迹中提取所有的基本块。如果所有的基本块被用于分析,则基本块中包含一些操作系统中的二进制文件执行块。此外,从恶意软件样本中提取的基本块中可能包括许多无意义的块。其结果是,安全分析人员需要分析的基本块数量增加,区分恶意软件特征就变得更加困难。此外,两个恶意软件样本的基本块之间的比较数量也急剧增加。与此相反,如果可以减少不必要块的数量,则可以尽可能地减少恶意软件分析时间,不仅可以减少单个恶意软件分析时间成本,还可以大大减少恶意软件样本数量。因此,我们从所有基本块中选择具备相关的可疑行为和功能的一些块。

如图 4 所示,主要块包括 CALL 指令、用于调用 API 的库函数、以及其它用户定义的函数。这是因为,用户定义的函数和各种系统调用用于执行大多数程序的行为和功能。如果块包含这些函数调用指令,则可以在恶意软件分析时提取恶意软件特征[2]。通过主要块选择技术,可以减少图像矩阵生成时间。

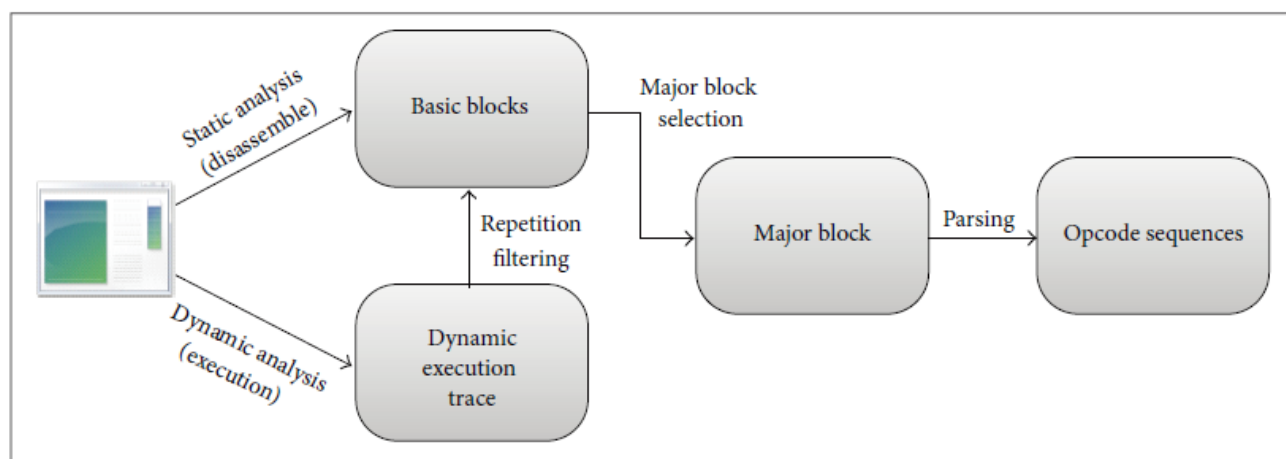


图 2：操作码序列的提取过程

| | | | |
|----------|------|-----|------------------------|
| 0042A68B | Main | JBE | SHORT Exploit_0042A69C |
| 0042A68D | Main | MOV | AL, BYTE PTR DS: [EDX] |
| 0042A68F | Main | INC | EDX |
| 0042A690 | Main | MOV | BYTE PTR DS: [EDI], AL |
| 0042A692 | Main | INC | EDI |
| 0042A693 | Main | DEC | ECX |
| 0042A694 | Main | JNZ | SHORT Exploit_0042A68D |
| 0042A68D | Main | MOV | AL, BYTE PTR DS: [EDX] |
| 0042A68F | Main | INC | EDX |
| 0042A690 | Main | MOV | BYTE PTR DS:[EDI],AL |
| 0042A692 | Main | INC | EDI |
| 0042A693 | Main | DEC | ECX |
| 0042A694 | Main | JNZ | SHORT Exploit_0042A68D |
| 0042A68D | Main | MOV | AL, BYTE PTR DS: [EDX] |
| 0042A68F | Main | INC | EDX |
| 0042A690 | Main | MOV | BYTE PTR DS: [EDI], AL |
| 0042A692 | Main | INC | EDI |
| 0042A693 | Main | DEC | ECX |
| 0042A694 | Main | JNZ | SHORT Exploit_0042A68D |
| 0042A68D | Main | MOV | AL, BYTE PTR DS: [EDX] |
| 0042A68F | Main | INC | EDX |
| 0042A690 | Main | MOV | BYTE PTR DS: [EDI], AL |
| 0042A692 | Main | INC | EDI |
| 0042A693 | Main | DEC | ECX |
| 0042A694 | Main | JNZ | SHORT Exploit_0042A68D |
| 0042A696 | Main | JMP | Exploit_0042A5FE |
| 0042A5FE | Main | ADD | EBX, EBX |

图 3：重复指令序列的例子

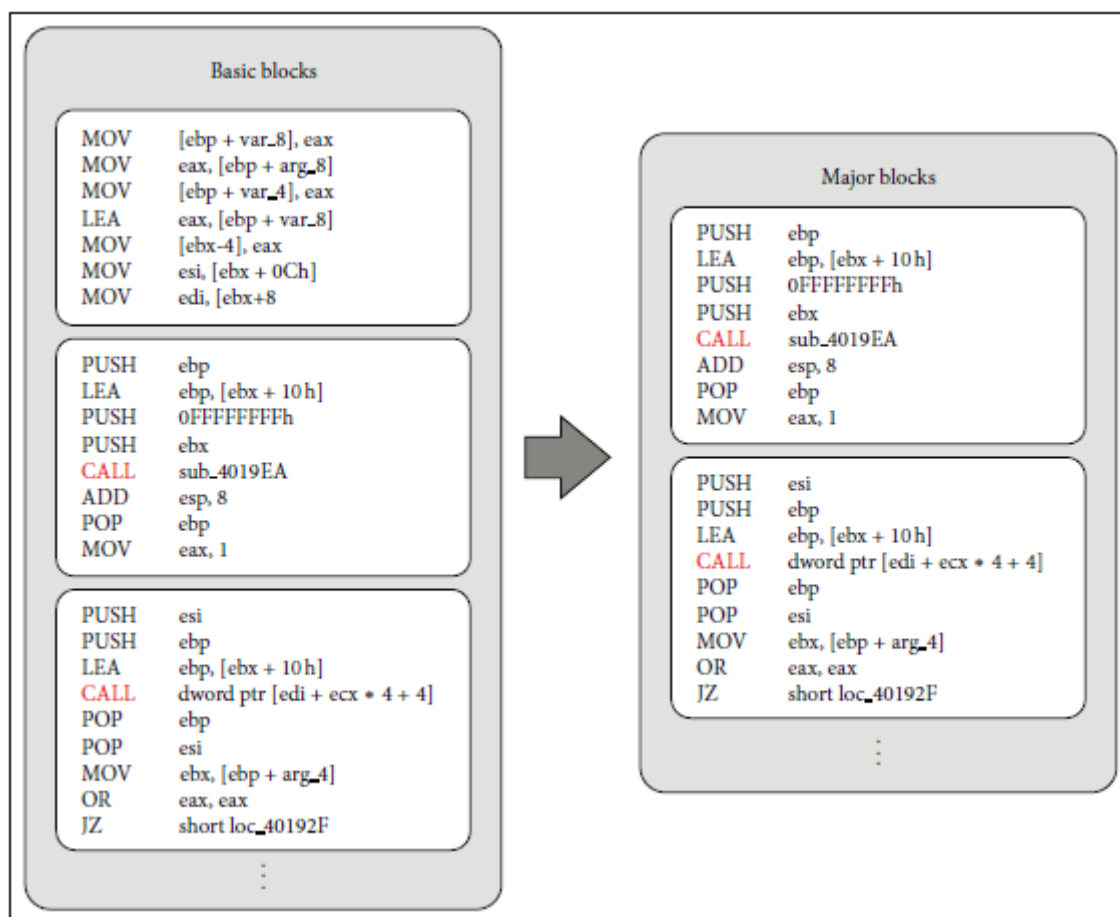


图 4：主要块选择

3.2.3 操作码序列提取。为了提取恶意软件特征，如图 5 所示，各个主要块的操作码序列被用作恶意软件信息。只有每个操作码的前 3 个字符被用于生成块的信息。使用 3 字符操作码的原因如下。从用于 Intel X86 汇编语言的整个操作码集合中，41.4%具有 3 个字符，这类操作码在二进制文件中的出现频率比其它操作码更高。另一方面，28.8%的操作码具有 4 个字符，17.8%具有 5 个字符，5.2%具有超过 6 个字符。因此，它们的出现频率相对较低。此外，即使被减少到 3 个字符，各个操作码的含义仍被保持，所以不同的操作码也可被区分开来。例如，4 个字符的操作码，如 PUSH，被减少到 3 个字符 PUS；或两个字符的操作码 OR 通过添加一个空字符增加为 3 个字符。然后，这些 3 个字符的操作码被连接到一起，并且字符串将块表示为操作码序列，操作码序列在下一步骤中用于生成图像矩阵的像素。

3.3 图像矩阵的生成。图 6 展示了步骤 2（将操作码序

列转化为图像矩阵的像素）的过程。散列函数用于决定像素的 X-Y 坐标和 RGB 颜色。

为了将二进制文件可视化图像矩阵，图像矩阵的长度和宽度都被初始化为 2^n ，其中 n 由用户选定。为了减少散列函数的冲突概率， n 应该足够大。在我们的实验中，我们选择了 n 等于 8，以减少冲突。

坐标定义模块和 RGB 颜色定义模块被用于生成图像矩阵。首先，坐标定义模块定义关于每个代码块的图像矩阵的像素的 (x, y) 坐标。其次，RGB 颜色定义模块定义了图像矩阵的像素的颜色值。RGB 颜色是通过计算红、绿和蓝色的 8 位值来定义的。

SimHash[38]被应用于步骤 1 的操作码提取，以确定像素的坐标和颜色值。SimHash 是局部敏感哈希函数，用于相似语句检测系统，它假定：如果输入值是相似的，则输出值也是相似的。也就是说，由于 SimHash 将输入字符串令牌化并为每个令牌生成哈希

值。如果两个输入字符串中的一些令牌是不同的，则生成的哈希值不会完全不同，而是相似的。

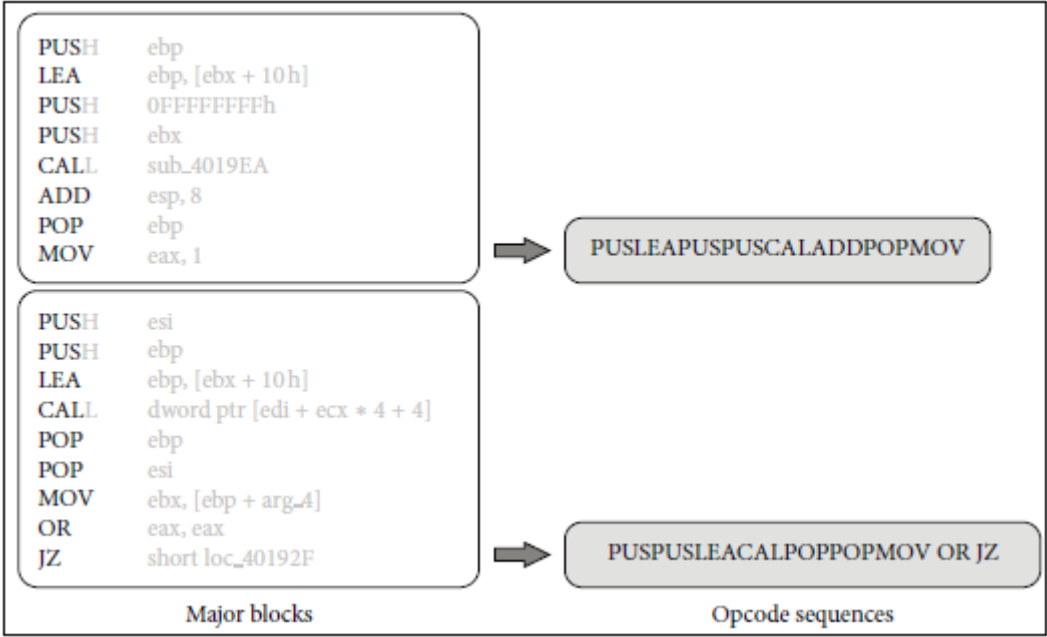


图 5：用作恶意软件信息的操作码序列

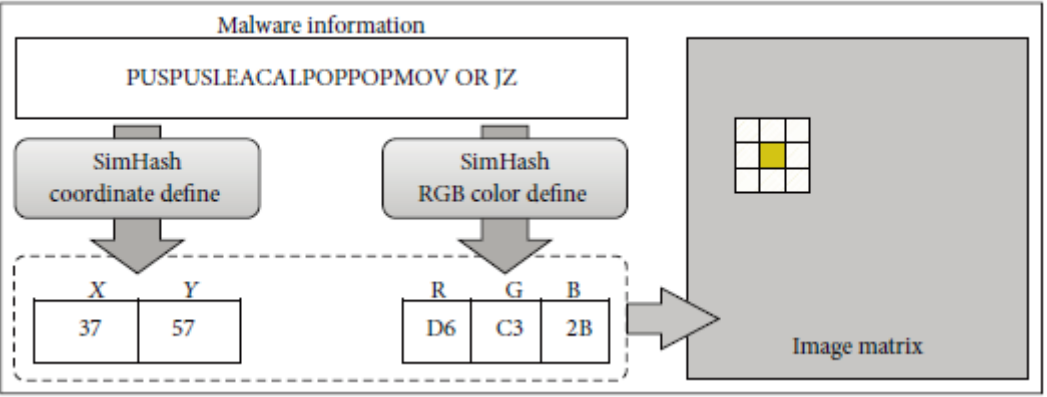


图 6：用操作码序列生成图像

因此，如果操作码序列的字符串相似，那么输出将是相似的，它们将映射到图像矩阵中的相似坐标上。

一旦各个像素的坐标和 RGB 颜色确定，RGB 颜色图像将被记录在图像矩阵的各个坐标上。为了使得分析人员更加方便地进行可视化分析，相应坐标的像素被同时记录下来。如图 7 所示，一个块从 (x-1, y-1) 到 (x+1, y+1) 坐标的 9 个像素被记录。

如果多个操作码序列的定义坐标是相邻的，导致图像相互重叠，如图 8 所示，则将 RGB 颜色的总和作为新像素的颜色。如果颜色总和超过 255 (0xFF)，结果将被设置为 255。例如，如果 RGB₁ 是 (255, 0, 0)，RGB₂ 是 (0, 176, 50)，则新的颜色则是 (255, 176, 50)。

根据主要块，图像矩阵上记录的像素的数量会有所不同。并且随着图像数量的增加，重叠的像素的数量也会增加。如果有太多的重叠的图像，那么就需要增加图像矩阵的大小。

| | | |
|------------|----------|------------|
| $x-1, y-1$ | $x, y-1$ | $x+1, y-1$ |
| $x-1, y$ | x, y | $x+1, y$ |
| $x-1, y+1$ | $x, y+1$ | $x+1, y+1$ |

图 7：一个操作码序列的九个像素

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | | |
| 1 | 1 | 1 | | |
| 1 | 1 | 3 | 2 | 2 |
| | | 2 | 2 | 2 |
| | | 2 | 2 | 2 |

$$\begin{aligned}
 R_3 &= \min(R_1 + R_2, FF) = \min(FF + 00, FF) = FF \\
 G_3 &= \min(G_1 + G_2, FF) = \min(FF + B0, FF) = B0 \\
 B_3 &= \min(B_1 + B_2, FF) = \min(FF + 50, FF) = 50 \\
 \therefore RGB_3 &= (FF, B0, 50)
 \end{aligned}$$

图 8：记录重叠像素的方法

3.4 代表性图像矩阵提取。因为每个恶意软件家族中存在许多恶意软件变种，随着恶意软件样本数量的增加，相似度计算的总量和时间也会增加。因此，我们提取每个恶意软件家族的代表性图像矩阵，以减少恶意软件的相似度计算的复杂度。也就是说，当一个新的恶意软件的样本被发现时，我们将其图像矩阵与恶意软件家族代表性图像矩阵进行对比，而非将其与现有恶意软件样本的所有图像矩阵进行对比，从而减少相似度计算的时间。

如图 9 所示，为了提取每个恶意软件家族的代表性图像矩阵，我们首先要生成恶意软件家族中的样本的图像矩阵。然后，通过只记录具有相同坐标和 RGB 颜色的图像矩阵，从而提取代表性图像矩阵。图 9 展示了生成恶意软件家族的代表性图像矩阵的例子。

3.5 利用图像矩阵计算相似度。使用图像矩阵进行相似度计算的优点是比使用字符串类型的操作码序列的精确匹配更快速，即使散列冲突会造成一些额外的误报。当使用字符串时，由于发现完全匹配的字符串对的过程，时间复杂度被定义为 $O(n^2)$ 。然而，如果图像矩阵被用于计算相似度，因为操作码序列的坐标和颜色都是通过 SimHash 定义的，则寻找字符串对的过程被跳过。因此，使用图像矩阵的相似度计算的时间复杂度被定义为 $O(n)$ ，因为只有记录在两个图像矩阵的相同坐标上的像素颜色信息用于计算图像矩阵之间的相似度。

首先对每个图像矩阵中的像素执行像素相似度计算。在这种情况下，相似度计算的最重要的考虑因素是：只有记录各个像素矩阵中的 RGB 颜色的像素应该被使用。图像矩阵在黑色背景的正方图像上有 RGB 颜色像素。如果黑色像素也被用于相似度计算，则不同恶意软件家族样本之间的相似度可能非常高。因此，当计算图像矩阵的相似度时，两个图像矩阵中相同坐标的像素的下述情况应该被考虑，如图 10 所示。在这种情况下，基于向量角度的距离测量算法用于计算颜色像素之间的相似度。此算法通过将构成各个图像的颜色像素表示为 3D 向量来计算相似值，如图 (1) 所示，然后使用角度和大小信息：

(a) 情况 1：如果两个图像矩阵中的区域的所有像素都是黑色，则不会进行像素相似度计算，并选择下一个像素。

(b) 情况 2 如果选定区域中的一个像素为黑色，且其他图像中的相应像素是彩色的，则像素相似度将被定义为 0。

(c) 情况 3：如果两个像素不是黑色而是彩色，则采用基于向量角度的距离测量算法来计算颜色像素相似度[39]，如下所示：

$$\delta(x_i, x_j) = \left[1 - \frac{2}{\pi} \cos^{-1} \left(\frac{x_i \cdot x_j}{|x_i| |x_j|} \right) \right] \left[1 - \frac{|x_i - x_j|}{\sqrt{3 \cdot 255^2}} \right]. \quad (1)$$

当考虑各种情况时，图像矩阵的相似度值的计算利用 (2) 所示的像素相似度计算的结果。即情况 3 中所计算的像素相似度值的总和除以情况 2 和 3 中计算出的像素的数量，以计算平均值：

$$\text{Sim}(A, B) = \frac{\text{sum of pixel similarity values in case 3}}{\# \text{ of pixels in case 2 and case 3}}. \quad (2)$$

4. 实验结果

4.1 实验数据和环境。使用本文中的可视化分析工具和表 1 中所示的恶意软件样本，生成图像矩阵，并计算相似度。首先，集合 A 包括来自 16 个家族的 290 个恶意软件样本，这些样本没有采用检测避免技术，例如模糊和压缩。使用反汇编器和静态分析，从这些恶意软件样本中提取基本块。第二，集合 B 包括来自 14 个家族的 560 个恶意软件样本，其中包括压缩和非压

缩的恶意软件样本。在动态分析环境中,我们通过 PIN 工具利用这些恶意软件样本生成动态执行痕迹,然后

通过重复过滤技术从动态执行痕迹中提取过滤后的基本块,如第 3.2.1 节所述。

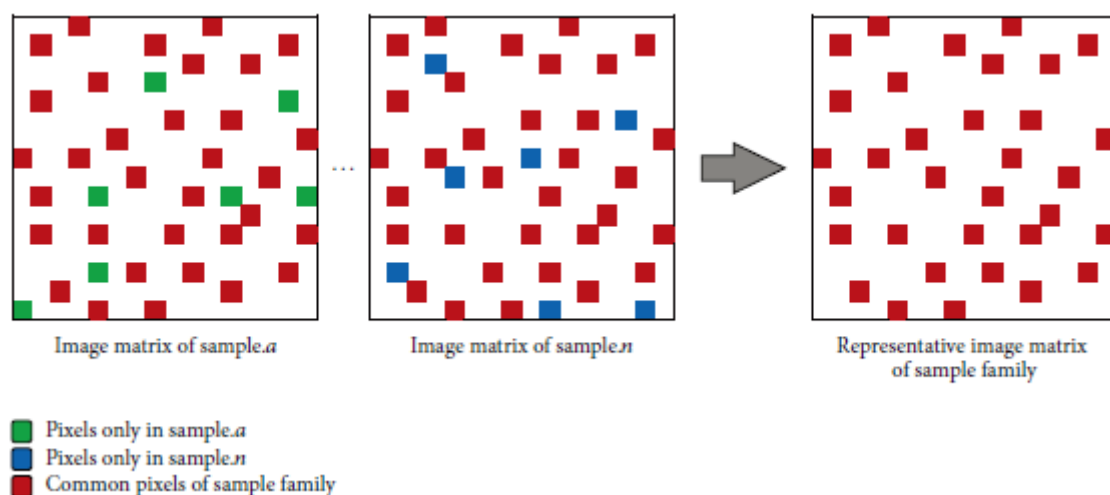


图 9：代表性图像矩阵的提取

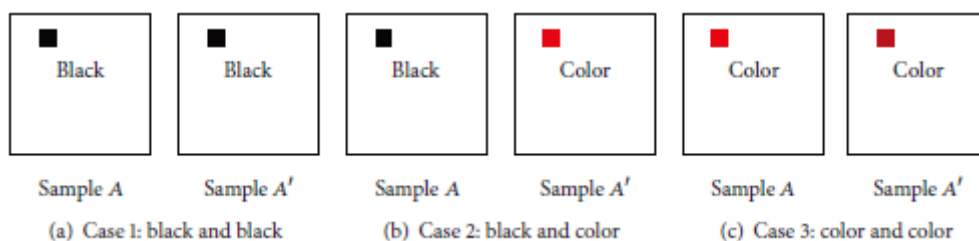


图 10：像素相似度计算中考虑的 3 种情况

在实验中,我们构建了一个实验环境,包括分析服务器、恶意软件服务器和监控机,如图 11 所示。我们在分析服务器中设置了 VMware vSphere ESXi 5.1,它具有 Intel Xeon E5-1607 处理器和 24GB 的主内存,我们安装了两个 Windows 操作系统作为客户机操作系统。在第一个 Windows 操作系统中,动态执行痕迹通过 PIN 提取。在另一个 Windows 操作系统中,则通过可视化分析工具来生成图像矩阵并计算相似度。提供给分析服务器的恶意软件样本和从分析服务器中提取的动态执行痕迹被存储在恶意软件服务器中。监控机通过 PowerCLI 工具(远程命令行接口)来控制分析服务器。

4.2 静态分析实验。对于此部分的实验,我们反汇编了集合 A 中的恶意软件样本,并从基本块提取了主要块。然后,我们利用这些主要块的操作码序列生成了图像矩阵并分析它们之间的相似度。

4.2.1 图像矩阵生成。在本文的实验中,我们将生成的图像矩阵的大小设置为 256×256 像素。如表 2 所示,采用这种图像矩阵大小的原因可简要概括为文件大小、

相似度计算时间以及分类准确度之间的折衷。通过(3)计算准确度:

$$\text{Accuracy} = \frac{\text{\# of correctly classified malware samples}}{\text{\# of total malware samples}} \quad (3)$$

图 12 显示了集合 A 中各个家族的恶意软件样本所生成的图像矩阵的例子。这只包括每个恶意软件家族的 3 个图像矩阵和 1 个代表性图像矩阵(通过记录那些普遍存在于所有图像矩阵的像素而提取出代表性图像矩阵)。由于用作恶意软件信息操作码序列的数量改变,记录在图像矩阵的像素的数量也会不同。在恶意软件的情况下,许多相同或相似的 RGB 颜色像素出现于同意家族的恶意软件样本的图像矩阵中。然而,即使像素被记录在不同图像矩阵的相同坐标上,如果相关像素的 RGB 颜色信息不同,像素的相似度也会有不同的值。我们的结果表明,同一恶意软件家族中的变种的图像矩阵是相似的,而不同家族的恶意软件样本的图像矩阵则明显不同。

图 13 显示了采用了主要块选择技术前后的图像矩

阵的差异。图像级数表示记录在图像矩阵中的像素的数量减少，原因是从基本块中选择主要块。采用主要块选择技术后相似度发生变化，这将在下节中介绍。

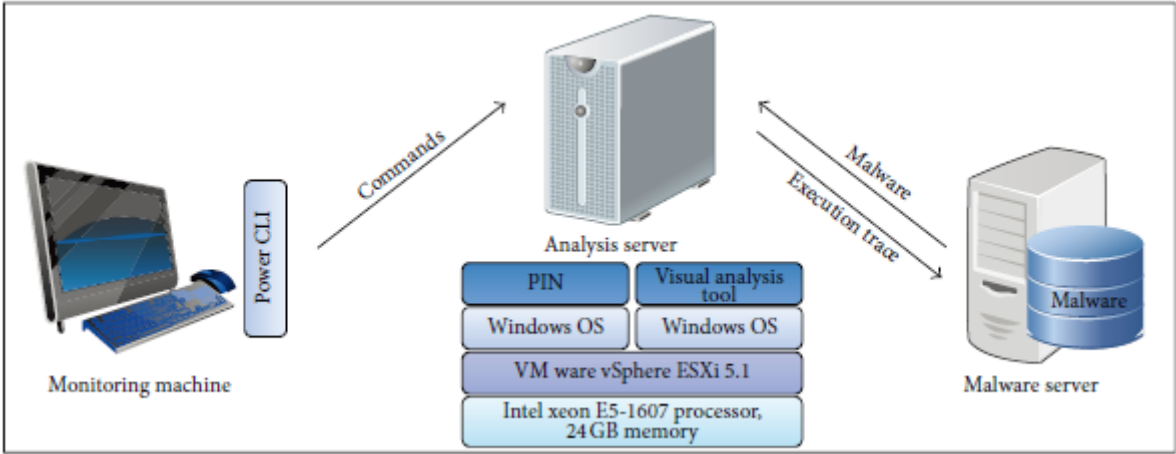


图 11：实验环境

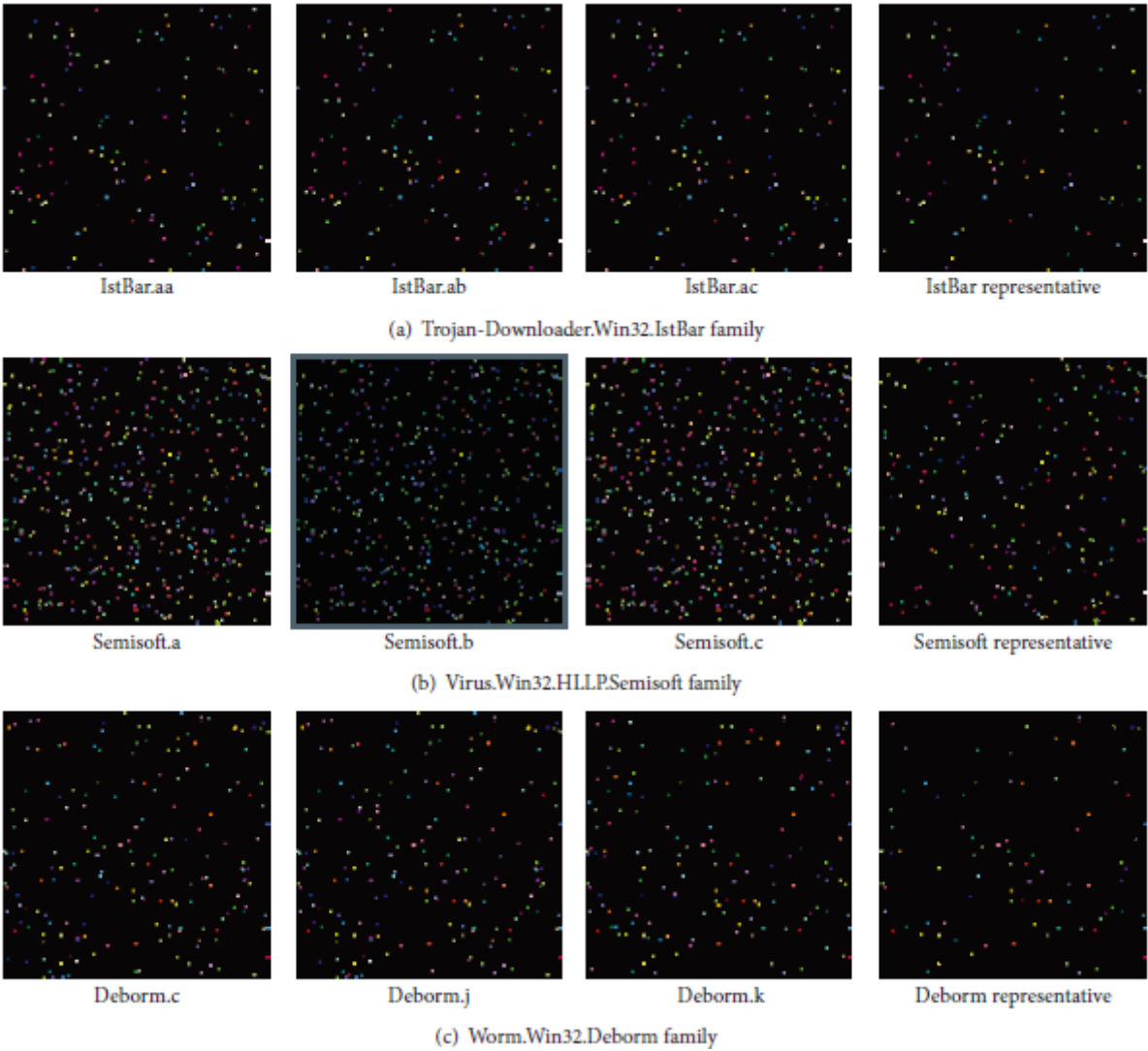


图 12：恶意软件家族样本的图像矩阵

表 1：恶意软件样本

| Set | Type | Family | Number of variants |
|-----|-------------------|-------------------|--------------------|
| A | Email-Worm | Klez | 9 |
| | Trojan-DDos | Boxed | 27 |
| | | IstBar | 41 |
| | | Ladder | 5 |
| | Trojan-Downloader | Lemmy | 26 |
| | | Mediket | 43 |
| | | OneClickNetSearch | 11 |
| | Trojan-Dropper | Tab | 8 |
| | | Eva | 6 |
| | | Evol | 3 |
| | | Fosforo | 4 |
| | Virus | Gpcode | 35 |
| | | Halen | 7 |
| | | Semisoft | 14 |
| | | Zepp | 11 |
| | Worm | Deborm | 40 |
| | Backdoor | Agobot | 40 |
| | | Bifrose | 40 |
| | | IRCBot | 40 |
| | | SdBot | 40 |
| | Trojan | Dialer | 40 |
| | | StartPage | 40 |
| B | Trojan-Downloader | Banload | 40 |
| | | Dyfuca | 40 |
| | | Swizzor | 40 |
| | | Bancos | 40 |
| | Trojan-Spy | Banker | 40 |
| | | Bagle | 40 |
| | Email-Worm | Kelvir | 40 |
| | P2P-Worm | SpyBot | 40 |

表 2：图像矩阵大小的选择

| Size (resolution) | File size (KB) | Similarity calculation time (ms, avg.) | Classification accuracy (avg.) |
|-------------------|----------------|--|--------------------------------|
| 128 × 128 | 48 | 5.3 | 0.9595 |
| 256 × 256 | 192 | 18.2 | 0.9814 |
| 512 × 512 | 768 | 66.4 | 0.9929 |

4.2.2 主要块选择。采用主要块选择后的图像矩阵相似度的计算如图 14 和 15 所示。当采用主要块选择技术后，同一家族中恶意软件样本的相似度变化的范围是从 0.002 (Tab 家族) 到 0.147 (Lemmy 家族)。不用家族的样本的相似度变化范围是 0.001 (Eva 家族) 到 0.053 (Klez 家族)。其结果是，同一家族中的恶意软件样本之间的相似度值大于 0.6，而不同家族中的恶意

软件样本之间的相似度的值在 0.1 以下。因此，虽然采用主要块选择技术会导致相似度值的变化，但是可以减少图像矩阵的生成时间，并且可以发现相似度值的明显差异。

表 3：任意选择的未知恶意软件样本

| Number | Malware sample | Most similar family (similarity) | |
|--------|---------------------|----------------------------------|---------|
| 1 | Klez.j | Klez | (0.181) |
| 2 | Boxed.g | Boxed | (0.190) |
| 3 | IstBar.gvf | IstBar | (0.302) |
| 4 | Ladder.f | Ladder | (0.339) |
| 5 | Lemmy.z | Lemmy | (0.341) |
| 6 | Mediket.ec | Mediket | (0.325) |
| 7 | OneClickNetSearch.k | OneClickNetSearch | (0.268) |
| 8 | Tab.gd | Tab | (0.348) |
| 9 | Eva.g | Eva | (0.317) |
| 10 | Evol.c | Evol | (0.329) |
| 11 | Fosforo.d | Fosforo | (0.341) |
| 12 | Gpcode.x | Gpcode | (0.306) |
| 13 | Halen.2619 | Halen | (0.352) |
| 14 | Semisoft.n | Semisoft | (0.281) |
| 15 | Zepp.d | Zepp | (0.265) |
| 16 | Deborm.ai | Deborm | (0.339) |
| 17 | Agobot.02.a | Mediket | (0.042) |
| 18 | SdBot.04.a | Boxed | (0.054) |

4.2.3 代表性图像矩阵的提取。对于这个实验，我们选择了集合 A 之外的任意的未知恶意软件样本。然后，我们分析该未知样本的图像矩阵与各个家族中的所有 290 个图像矩阵和 16 个代表性图像矩阵之间的相似度计算时间和相似度值。

图 16 示出了该未知样本与各个家族所有恶意软件样本的图像矩阵及代表性图像矩阵的相似度计算的结果。当使用所有的图像矩阵时，Tab 家族的平均相似度值为 0.781，而所有其他家族的相似度值则小于 0.05。当使用各个家族的代表性图像矩阵时，Tab 家族的平均相似度值是 0.348，而其它家族的相似度值则小于 0.03。因此，未知样本有望成为 Tab 家族的一个变种。事实上，用于本实验的未知样本的诊断名称为 Trojan-Dropper.Win32.Tab.gd.。

表 3 显示了本实验所使用的未知恶意软件样本的列表，包括使用代表性图像矩阵的相似度计算的结果。除了 Agobot.02.a 和 Sdbot.04.a，这些恶意软件样本被检测为集合 A 中的各个家族的变种，而代表性图像矩

阵的相似度值的范围是 0.181 到 0.352。由于 Agobot.02.a 和 Sdbot.04.a 样本不是集合 A 中恶意软件家族的变种，相比于现有各个家族的代表性图像矩阵，它们的相似度值非常低。

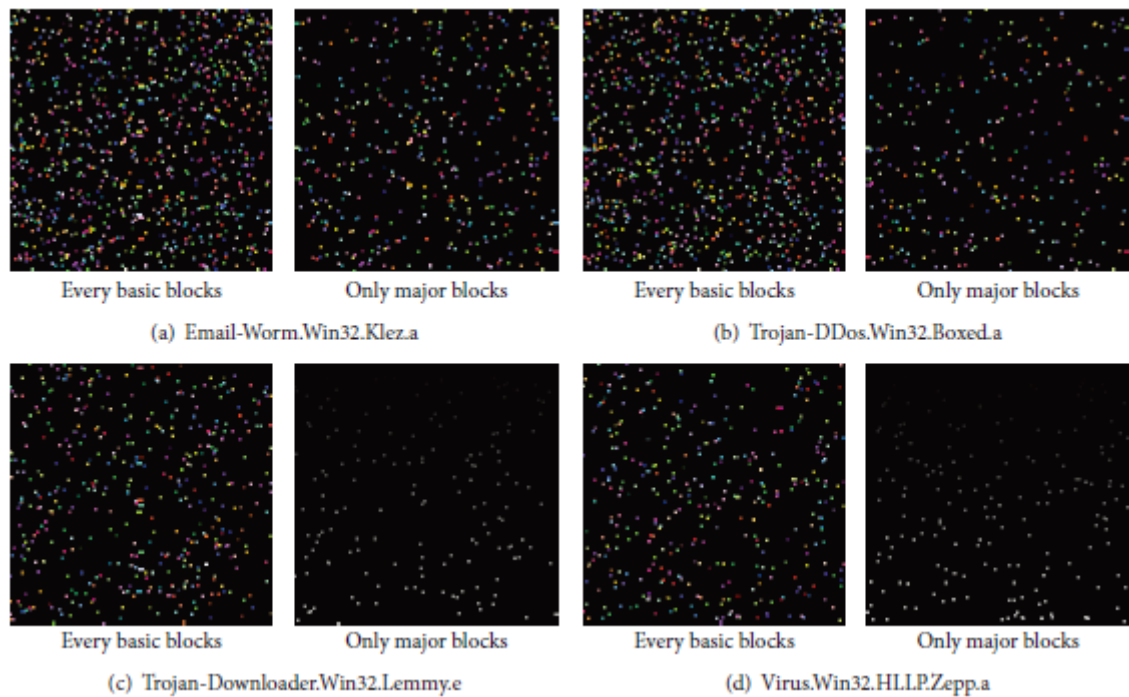


图 13：使用和不使用主要块选择时的图像矩阵对比



图 14：采用主要块选择技术后，各个家族的恶意软件样本的图像矩阵的相似度计算。（左）

图 15：采用主要块选择技术后，不同家族的恶意软件样本的图像矩阵的相似度计算。（右）

4.2.4 恶意软件分类的可行性。图 17 显示了采用我们提出的所有方法（即，主要块选择技术和代表性图像矩阵提取技术）后的相似度值的变化。同一家族的恶意软件样本之间的相似度值界于 0.19 和 0.36 之间，而不

同家族的恶意软件样本之间的相似度均小于 0.05。分类准确度（使用通过静态分析生成的图像矩阵得到）是 0.9896。也就是说，集合 A 中只有 3 个恶意软件样本被错误分类为其他恶意软件家族。

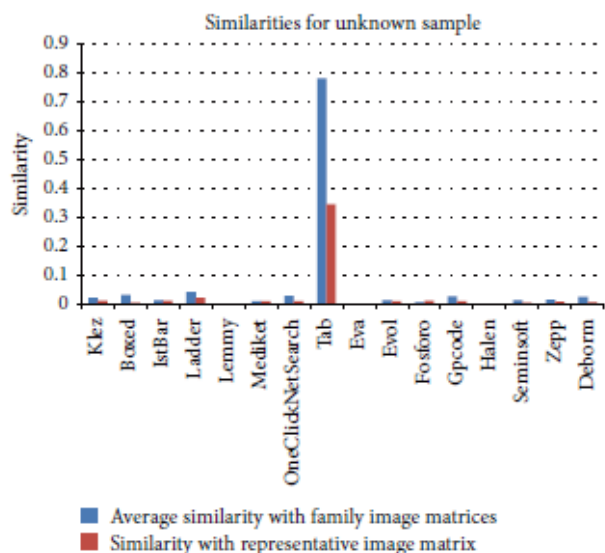


图 16：未知样本与各个家族的代表性图像矩阵的相似度值。

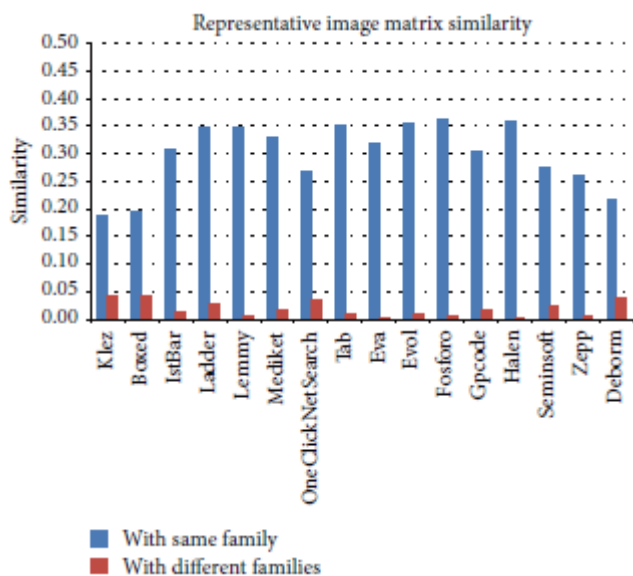


图 17：使用 3 个提出的技术得到的相似度计算结果。

我们得到的结果稍好于使用二进制结构分析[30]的平均分类准确度 0.9757。因此，我们得出结论，我们的方法对于恶意软件分类是可行的，因为相比于来自不同家族的恶意软件样本，同一家族的恶意软件样本之间的相似度会比较高。

4.3 基于执行痕迹的实验。对于基于执行痕迹的实验，我们使用 PIN 工具在动态分析环境中执行集合 B 中的恶意软件样本（如表 1 所示）。然后生成动态执行痕迹，之后从这些执行痕迹中提取需进行经过重复过滤的基本块。过滤后，选择与可疑行为和功能有关的主要块。我们提出的技术被应用到这些执行痕迹中，以生成图像矩阵并分析相似度。

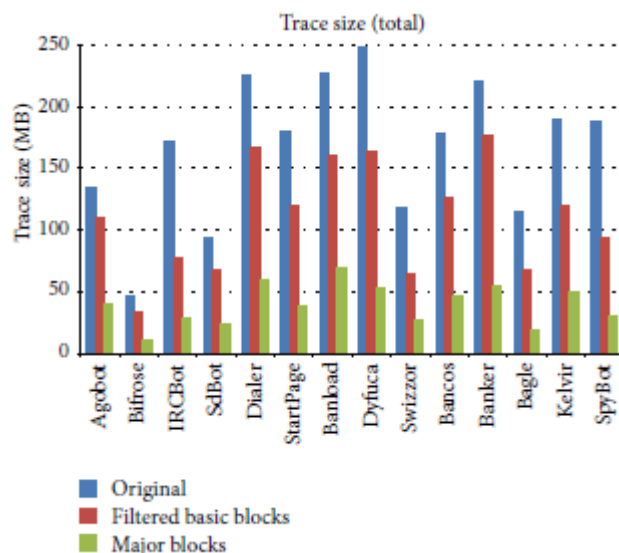


图 18：下面的重复过滤和重大块选择应用改变执行跟踪的大小。

图 18 显示了采用重复过滤和主要块选择技术之后执行痕迹的减少。如果采用重复过滤技术后，执行痕迹第一次减少，然后又采用了主要块选择技术，那么相比于初始执行痕迹，执行痕迹的平均减少量为 76.5%（最低 69.3%，最高 83.6%）。

图 19 显示了采用重复过滤和主要块选择技术之后生成的图像矩阵的变化。当比较 3 个图像矩阵时，我们发现记录在图像矩阵中的像素的数量减少了。

图 20 和图 21 显示了采用重复过滤和主要块选择技术之后相似度值的变化。虽然变化值都不大，但是如果正确设置相似度阈值，那么有些恶意软件家族还是能够予以区分的。

在这些实验中，同一家族的恶意软件样本的平均相似度值大约是 0.65，不同家族的恶意软件样本的平均相似度值大约是 0.36。相比于先前所介绍的静态分析结果，基于执行痕迹的实验结果显示出了相对较小的差异。这些结果的原因是：当每个家族的恶意软件样本在动态环境中执行来提取动态执行痕迹时，相似的系统动态链接库（DLL）就会被调用。其结果是，由于 DLL 调用和其在动态执行痕迹中的执行，相似的操作码序列被记录在各个家族的图像矩阵中，所以相似度值会增加。然而，使用图像矩阵（根据执行痕迹而生成），集合 B 中只有 15 个恶意软件样本被错误分类，而这种结果类似于[30]中的准确性。

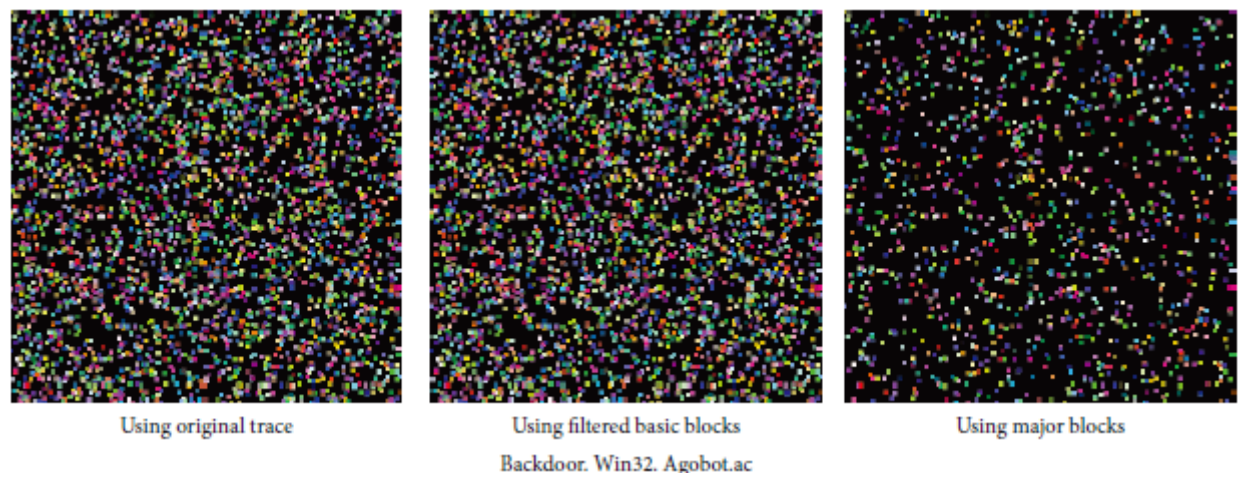


图 19：采用重复过滤和主要块选择技术之后生成的图像矩阵的变化

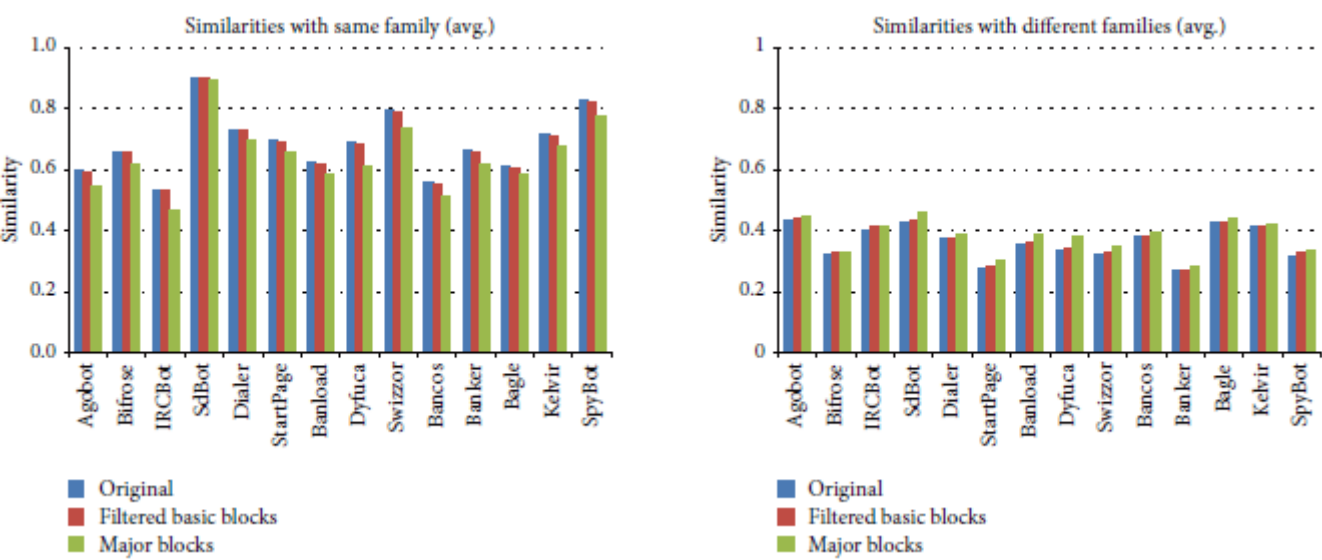


图 20：采用重复过滤和主要块选择技术之后，同一家族中样本的相似度值的变化。（左）

图 21：采用重复过滤和主要块选择技术之后，不同家族中样本的相似度值的变化。（右）

5. 结论和未来工作

在本文中，我们提出了一种新颖的方法，通过生成图像矩阵来可视化地分析恶意软件样本。为了生成图像矩阵，我们通过静态分析和动态分析来提取操作码序列。此外，我们使用图像矩阵中的 RGB 颜色像素的向量化值来计算恶意软件变种之间的相似度。使用图像矩阵的相似度计算方法比使用字符串类操作码序列或基本块的精确匹配更加快速。我们所提出的方法作为一种可视化分析工具。实验结果表明，同一家族的恶意软件样本转换为图像矩阵后是相似的，而恶意

软件变种之间的相似度更高。采用我们的方法，安全分析人员可以直观地分析恶意软件样本，并可以区分相似的恶意软件样本作进一步分析。我们未来的研究包括使用基于 GPGPU 的并行技术和实时处理的更快速的恶意软件检测和分类。

利益冲突

本文作者宣布，关于本文的发表不存在任何利益冲突。

致谢

该研究项目在 2013 年获得文化、体育和旅游部 (MCST) 和韩国著作权委员会的支持。本文是根据发表于《适应和收敛系统研究 2013》(RACS'2013) 的第一版本所修订的[40]。

参考文献

- [1] M. Christodorescu and S. Jha, "Testing malware detectors," in Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '04), pp. 34–44, July 2004.
- [2] B. Kang, T. Kim, H. Kwon, Y. Choi, and E. G. Im, "Malware classification method via binary content comparison," in Proceedings of the ACM Research in Applied Computation Symposium (RACS '12), pp. 316–321, San Antonio, Tex, USA, October 2012.
- [3] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC '07), pp. 421–430, Miami Beach, Fla, USA, 2007.
- [4] M. D. Ernst, "Static and dynamic analysis: synergy and duality," in Proceedings of the ICSE Workshop on Dynamic Analysis (WODA '03), pp. 24–27, Citeseer, 2003.
- [5] S. Cesare and Y. Xiang, "A fast flowgraph based classification system for packed and polymorphic malware on the endhost," in Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA '10), pp. 721–728, Perth, Australia, April 2010.
- [6] G. Bonfante, M. Kaczmarek, and J.-Y. Marion, "Architecture of a morphological malware detector," Journal in Computer Virology, vol. 5, pp. 263–270, 2009.
- [7] I. Briones and A. Gomez, "Graphs, entropy and grid computing: automatic comparison of malware," in Proceedings of the Virus Bulletin Conference (VB '08), pp. 1–12, Ottawa, Canada, October 2008, <http://pandalabs.pandasecurity.com/blogs/images/PandaLabs/2008/10/07/IsmaelBriones-VB2008.pdf>.
- [8] S. Shang, N. Zheng, J. Xu, M. Xu, and H. Zhang, "Detecting malware variants via function-call graph similarity," in Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE '10), pp. 113–120, Nancy, France, 2010.
- [9] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," Journal in Computer Virology, vol. 7, no. 4, pp. 233–245, 2011.
- [10] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in Proceedings of the ACM SIGKDD Workshop on Cyber Security and Intelligence Informatics (CSI-KDD '09), pp. 23–31, ACM, June 2009.
- [11] B. B. Rad and M. Masrom, "Metamorphic virus variants classification using opcode frequency histogram," in Proceedings of the 14th WSEAS International Conference on Computers, pp. 147–155, Corfu Island, Greece, July 2010.
- [12] D. Bilar, "Opcodes as predictor for malware," International Journal of Electronic Security and Digital Forensics, vol. 1, pp. 156–168, 2007.
- [13] K. S. Han, S.-R. Kim, and E. G. Im, "Instruction frequency-based malware classification method," Information, vol. 15, no. 7, pp. 2973–2984, 2012.
- [14] I. Santos, F. Brezo, J. Nieves et al., "Idea: Opcode-sequence-based malware detection," in Engineering Secure Software and Systems, pp. 35–43, Springer, 2010.
- [15] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables (SAVE)," in Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04), pp. 326–334, December 2004.
- [16] A. Walenstein, M. Venable, M. Hayes, C. Thompson, and A. Lakhota, "Exploiting similarity between variants to defeat malware," in Proceedings of the BlackHat DC Conference, 2007.
- [17] G. Chowdhury, Introduction to Modern Information Retrieval, Facet publishing, 2010.
- [18] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. X. Song, "Dynamic spyware analysis," in Proceedings of the Usenix Annual Technical Conference, pp. 233–246, 2007.
- [19] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, "Synthesizing near-optimal malware specifications from suspicious behaviors," in Proceeding of the 31st IEEE Symposium on Security and Privacy (SP '10), pp. 45–60, Oakland, Calif, USA, May 2010.
- [20] P. Vinod, H. Jain, Y. K. Golecha, M. S. Gaur, and V. Laxmi, "Medusa: metamorphic malware dynamic analysis using signature from API," in Proceedings of the 3rd International Conference on Security of Information and Networks (SIN '10), pp. 263–269, ACM, September 2010.
- [21] Q. G. Miao, Y. Wang, Y. Cao, X. G. Zhang, and Z. L. Liu, "API Capture—a tool for monitoring the behavior of malware," in Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE '10), pp. V4-390–V4-394, August 2010.
- [22] K. S. Han, J. H. Lim, B. Kang, and E. G. Im,

“Malware analysis using visualized images and entropy graphs,” *International Journal of Information Security*, 2014.

[23] P. Trinius, T. Holz, J. Goebel, and F. C. Freiling, “Visual analysis of malware behavior using treemaps and thread graphs,” in *Proceedings of the 6th International Workshop on Visualization for Cyber Security (VizSec '09)*, pp. 33–38, Atlantic City, NJ, USA, October 2009.

[24] J. Saxe, D. Mentis, and C. Greamo, “Visualization of shared system call sequence relationships in large malware corpora,” in *Proceedings of the 9th International Symposium on Visualization for Cyber Security (VizSec '12)*, pp. 33–40, ACM, October 2012.

[25] G. Conti, E. Dean, M. Sinda, and B. Sangster, “Visual reverse engineering of binary and data files,” in *Visualization for Computer Security*, pp. 1–17, Springer, Berlin, Germany, 2008.

[26] B. Anderson, C. Storlie, and T. Lane, “Improving malware classification: bridging the static/dynamic gap,” in *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence (AISEC '12)*, pp. 3–14, Raleigh, NC, USA, October 2012.

[27] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, “Malware images: visualization and automatic classification,” in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, p. 4, ACM, 2011.

[28] A. Oliva and A. Torralba, “Modeling the shape of the scene: a holistic representation of the spatial envelope,” *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[29] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, “Context-based vision system for place and object recognition,” in *Proceedings of the 9th IEEE International Conference on Computer Vision*, pp. 273–280, IEEE, October 2003.

[30] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, “A comparative assessment of malware classification using binary texture analysis and dynamic analysis,” in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISEC '11)*, pp. 21–30, 2011.

[31] Y. Kang and A. Sugimoto, “Image categorization and semantic segmentation using scale-optimized textons,” *Journal of IT Convergence Practice*, vol. 2, pp. 2–14, 2014.

[32] C. Eagle, *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*, No Starch Press, 2008.

[33] O. Yuschuk, “Ollydbg,” 2007, <http://www.ollydbg.de>.

[34] Y. Wei, Z. Zheng, and N. Ansari, “Revealing packed malware,” *IEEE Security and Privacy*, vol. 6, no. 5, pp. 65–69, 2008.

[35] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, “PolyUnpack: automating the hidden-code extraction of unpack-executing malware,” in *Proceeding of the 22nd Annual Computer Security Applications Conference (ACSAC '06)*, pp. 289–300, Miami Beach, Fla, USA, December 2006.

[36] S. Berkowits, “Pin—A Dynamic Binary Instrumentation Tool,” 2012, <https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>.

[37] B. Kang, K. S. Han, B. Kang, and E. G. Im, “Malware categorization using dynamic mnemonic frequency analysis with redundancy filtering,” 2013.

[38] M. S. Charikar, “Similarity estimation techniques from rounding algorithms,” in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pp. 380–388, ACM, New York, NY, USA, 2002.

[39] D. Androutsos, K. N. Plataniotis, and A. N. Venetsanopoulos, “Novel vector-based approach to color image retrieval using a vector angular-based distance measure,” *Computer Vision and Image Understanding*, vol. 75, no. 1, pp. 46–58, 1999.

[40] K. S. Han, J. H. Lim, and E. G. Im, “Malware analysis method using visualization of binary files,” in *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, pp. 317–321, ACM, 2013.