doi:10.3969/j.issn.1000-436x.2014.08.016

基于纹理指纹的恶意代码变种检测方法研究

韩晓光[1]，曲武[2,3]，姚宣霞[1]，郭长友[1]，周芳[1]

（1. 北京科技大学 计算机与通信工程学院，北京 100083；
2. 北京启明星辰信息安全技术有限公司 核心研究院，北京 100193；
3. 清华大学 计算机科学与技术系，北京 100084）

摘 要：提出一种基于纹理指纹的恶意代码特征提取及检测方法，通过结合图像分析技术与恶意代码变种检测技术，将恶意代码映射为无压缩灰阶图片，基于纹理分割算法对图片进行分块，使用灰阶共生矩阵算法提取各个分块的纹理特征，并将这些纹理特征作为恶意代码的纹理指纹；然后，根据样本的纹理指纹，建立纹理指纹索引结构；检测阶段通过恶意代码纹理指纹块生成策略，采用加权综合多分段纹理指纹相似性匹配方法检测恶意代码变种和未知恶意代码；在此基础上，实现恶意代码的纹理指纹提取及检测原型系统。通过对 6 种恶意代码样本数据集的分析和检测，完成了对该系统的实验验证。实验结果表明，基于上述方法提取的特征具有检测速度快、精度高等特点，并且对恶意代码变种具有较好的识别能力。

关键词：网络安全；恶意代码变种检测；纹理指纹；空间相似性检索

中图分类号：TP309.5 **文献标识码**：A **文章编号**：1000-436X(2014)08-0125-12

Research on malicious code variants detection based on texture fingerprint

HAN Xiao-guang[1], QU Wu[2,3], YAO Xuan-xia[1], GUO Chang-you[1], ZHOU Fang[1]

(1. School of Computer&Communication Engineer, University of Science&Technology Beijing, Beijing 10083, China;
2. Core Research Institute, Beijing Venustech Cybervision Co. Ltd., Beijing 100193, China;
3. Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: A texture-fingerprint-based approach is proposed to extract or detect the feature from malware content. The texture fingerprint of a malware is the set of texture fingerprints for each uncompressed gray-scale image block. The malicious code is mapped to uncompressed gray-scale image by integrating image analysis techniques and variants of malicious code detection technology. The uncompressed gray-scale image is partitioned into blocks by the texture segmentation algorithm. The texture fingerprints for each uncompressed gray-scale image block is extracted by gray-scale co-occurrence matrix algorithm. Afterwards, the index structure for fingerprint texture is built on the statistical analysis of general texture fingerprints of malicious code samples. In the detection phase, according to the generation policy for malicious code texture fingerprint, the prototype system for texture fingerprint extraction and detection is constructed by employing the integrated weight method to multi-segmented texture fingerprint similarity matching to detect variants and unknown malicious codes. Experimental results show that the malware variants detection system based on the proposed approach has good performance not only in speed and accuracy but also in identifying malware variants.

Key words: network security; malware variants detection; texture fingerprint; spatial similarity retrieval

---

收稿日期：2014-03-10；**修回日期**：2014-04-30

基金项目：国家重点基础研究发展规划（“973”计划）基金资助项目（2007CB310803）；国家自然科学重点基金资助项目（61035004）；国家自然科学基金资助项目（60875029）

Foundation Items: The National Basic Research Program of China (973 Program) (2007CB310803); The National Natural Science Foundation of China, Key Program (61035004); The National Natural Science Foundation of China, General Program (60875029)

1 引言

随着互联网的蓬勃发展，恶意代码的规模呈指数级增长，已经成为威胁互联网安全的关键因素之一。Symantec 的 2010 年报告[1]指出，2008 年新增恶意代码特征标签 169 323 例，2009 年新增 2 895 802 例，2010 年 Symantec 的恶意代码语料库规模达到了 286 000 000 例。仅由 Symantec 的监测数据，已可见恶意代码数量的日益庞大及其威胁的日益严重。由于恶意代码检测技术的局限性，仍有大量恶意代码无法有效查杀。特别是，免杀的恶意代码及其变种不断出现，是恶意代码检测形势日益严峻的根本原因。Symantec 2012 年报告[2]指出，与 2011 年相比，2012 年移动终端恶意代码家族同比增长 58%，迄今为止，2013 年同比增长 59%。同时，恶意代码变种的数目急剧增长，从 2011 年，每个家族平均变种率为 5:1，2012 年飙升到 38:1。这表明，恶意代码作者花费更多的时间在轻微改动或打包用以进一步传播以及规避检测。因此，恶意代码变种检测问题是当前恶意代码防范的重点，同时也是难点。

当前的恶意代码变种在实现技术上大致可分为 2 大类：一类是共用基础技术（核心模块或理论），黑客通过重用基础模块实现恶意代码变种；另一类是特别针对现有防范和检测技术而研发的恶意代码混淆技术。恶意代码混淆技术按照其实现原理可分为 2 类[3]：一类是干扰反逆向（反汇编）的混淆，使反逆向不能够得到正确的分析结果，从而阻碍进一步机理分析；另一类是指令和控制流混淆，这类混淆技术通常采用加壳、垃圾代码插入、等价指令替换、寄存器重新分配及代码变换等方式，改变恶意代码的语法特征，隐藏其内部调用逻辑关系。通常，恶意代码检测是基于特征向量的，该特征向量标识了恶意代码的本质特征，良好的特征提取算法是变种检测的核心技术。目前，主流的恶意代码特征抽取算法主要分为 2 类：基于恶意代码二进制的静态特征和基于恶意代码运行时行为的动态检测。基于恶意代码静态特征的检测方法通过分析其 PE 文件结构、二进制字节码、反汇编后的代码、反汇编后系统调用等因素来获取恶意代码的静态特征，利用基于学习的分类算法区分良性软件与恶意代码，实现未知和已知的恶意代码检测。基于静态特征的恶意代码检测通常容易受代码混淆技术（加壳、变形、多态技术等）的影响，该技术提高了逆向难度，使其几乎很难逆向或是不可能（成本因素），而且静态检测方法没有真实地运行软件，判断是否为恶意的软件行为没有展现，误报和漏报都比较明显。基于动态特征的恶意代码检测方法主要原理是将待检代码程序放置在一个虚拟机环境（沙箱或蜜罐）中，通过检测目标程序运行过程的恶意行为来判断是否为恶意代码。动态检测方法又分为粗粒度方法和细粒度方法。粗粒度方法通过运行恶意代码分析其行为所对应 API 调用序列来进行恶意代码检测，细粒度方法通过恶意代码运行时的动态指令序列来进行检测。与静态检测方法相比，动态检测方法更为有效，无需考虑解分组、解密等复杂过程。然而，动态检测方法是时间密集型和资源消耗型的方法，虚拟机执行包括解分组、执行、全路径探索来捕捉调用序列、退出等过程，动态检测方法平均分析时间为 3~5 min，即使过程压缩到 30 s，2010 年 Symantec 的恶意语料库也需要花费 254 年处理一遍。因此，动态检测方法可扩展性不足。而且，由于激发条件不能满足，一些恶意代码的行为不能表现出来。

基于文献[4,5]的部分成果，结合恶意代码同源性分析方法，本文提出基于纹理指纹的恶意代码变种检测方法。同源恶意代码二进制序列是指从某一共同恶意代码祖先经趋异进化过程而形成的不同二进制序列。也就是说，为了规避查杀工具识别，恶意代码经过人为改写或程序自动异变，产生具有新特性的恶意代码。这种恶意代码在一定程度上也可以认为是一种新的恶意代码。通常，恶意代码变种包括 2 种变异方向：关键区变异和非关键区变异。关键区变异将导致一种新的恶意代码产生，它是恶意代码表现方式、行为方式、感染机理和发作条件等方面发生了实质性变化的一种变异现象。与之对应的，非关键区变异情况下，恶意代码并没有发生实质性的改变，例如仅在恶意代码文件内容中加入了一些“空指令(NOP)”等。非关键区变异并不会产生恶意代码，但它导致恶意代码的识别更为复杂，从而导致个别杀毒软件漏查。而无论是关键区变异还是非关键区变异，变种的恶意代码很大程度上都共用了祖先的大部分源代码，即与祖先恶意代码为同源恶意代码家族。

本文提出的基于纹理的恶意代码特征提取及检测方法，通过结合图像纹理分析技术与恶意代码

变种检测技术，将恶意代码映射为无压缩灰阶图片，基于纹理分割算法对图片进行分块，使用灰阶共生矩阵算法提取各个分块的纹理特征，并将这些纹理特征作为恶意代码的纹理指纹；然后，利用样本的纹理指纹，建立纹理指纹索引结构。在检测阶段，通过恶意代码分段纹理指纹生成策略，使用综合多分段纹理指纹相似性匹配算法检测未知恶意代码及其变种。基于图像映射的方法可以有效地避免反追踪、反逆向逻辑以及其他常用的代码混淆策略。而且，该方法能够有效地检测使用特定封装工具打包的恶意代码。本文的贡献主要包括如下 4 个部分。

1) 引入了一种基于图像映射的恶意代码描述方法。本方法与静态检测和动态检测机制完全不同，能够有效地克服反追踪、反逆向逻辑以及其他常用的代码混淆策略。因此，可显著提高恶意代码变种检测能力，同时降低了纹理特征库的规模，从而从根本上改进检测效率。

2) 提出了通过提取图像纹理特征值来计算恶意代码纹理指纹的方法，从代码同源映射为纹理特征向量最近邻距离，然后建立相应的纹理指纹匹配算法，解决了等价指令替换、垃圾代码插入等混淆代码的检测问题。

3) 为提高检测速度和精度，提出基于段自增长的纹理分割算法，检测过程中，使用加权综合多分段纹理指纹相似性匹配方法检测恶意代码变种和未知恶意代码。

4) 建立了基于内容指纹的恶意代码变种检测原型系统（MVDS-TF, malicious code variants detection system based on texture fingerprint），实现了基于灰阶共生矩阵的灰阶图像特征提取方法，并基于该特征实现了恶意代码纹理指纹的提取方法。使用 M-Trees 算法建立纹理指纹树形索引结构。

2　相关理论和技术

本文主要研究恶意代码分块纹理特征计算及变种检测算法。为了实现恶意代码二进制文件分块纹理特征提取及变种检测，首先要对主流的恶意代码检测技术进行简述。在此基础上，研究恶意代码二进制 PE 文件的纹理特征提取及描述方法，并基于提取的纹理特征及其描述，实现基于恶意代码分块纹理特征的检测。因此，本文从恶意代码检测、恶意代码二进制特征描述这 2 个方面介绍相关研究工作。

2.1　恶意代码检测

通常，主流的恶意代码检测技术可分为 2 类[6]：启发式检测方法和基于特征码的检测方法。启发式检测方法，例如 Rootkit Revealer[7]检测系统，通过比较系统上层信息以及来自内核的文件系统状态来识别隐藏的进程、文件和相关的注册表信息。对于启发式检测方法，由于其规则制定时严重依赖研究人员的经验，容易导致高误报率或漏报率。因此，该方法在学术领域研究较广，但网络安全公司实际应用不多。基于特征码的恶意代码检测方法根据提取恶意代码二进制文件的形态特征，通过模式匹配的方式检测恶意代码，该方法具有速度快、效率高、误报率低等优点，是网络安全公司常用的方法。在实际应用中，通常会遇到基于特征码的检测方法也不能识别特征的未知恶意代码，即恶意代码经过简单变形或者混淆即可免杀。因此，研究人员提出了基于行为特征的恶意代码检测方法，一定程度上能够降低代码混淆的影响，但该方法仍然不足以解决等价行为替换、系统调用重排等方法造成的漏报或误报问题。基于行为特征的检测从代码行为出发提取特征，例如，Kirda 等人[8]根据间谍代码获取用户敏感数据，再将数据泄露的行为特征进行检测，但该方法仅限于检测间谍类的恶意代码。基于语义的特征检测从语义角度抽取特征，例如 Christodorescu 等人[9]通过静态分析恶意代码，结合指令语义信息，使用三元操作符构造特征向量来进行检测，该方法可有效对抗指令重排、垃圾代码插入、寄存器重分配等混淆技术的干扰。但由于是指令级分析和提取，对行为层混淆技术的抗干扰能力较弱，且其匹配判定过程比较复杂。Kinder 等人[10]利用形式化方法，通过形式化模型来检测恶意代码。Sathyanarayan 等人[11]同样从语义角度出发，使用关键系统 API 调用之间的相互关系，静态分析挖掘一类代码的共同特征，采用统计学的方法进行检测，该方法可对抗部分代码混淆技术的干扰。然而，对等价系统调用替换等混淆方法未能很好地解决，且存在静态分析方法的局限性。结合指令层的污点传播分析与行为层的语义分析，王蕊等人[6]提出一种基于语义的恶意代码行为特征提取及检测方法。该方法提取恶意代码的关键行为及行为之间的依赖关系；然后，利用抗混淆引擎识别语义无关及语义等价行为，获取具有一定抗干扰能力的恶意代码行为特征。但该方法由于较差的时间性能，实用性不强。

根据检测时代码状态，恶意代码检测技术还可以分为静态分析和动态分析 2 种。静态分析首先对恶意代码二进制文件进行反汇编，并在此基础上提取代码的特征信息。静态分析可在无须实际执行代码的情况下，对代码进行全面的分析，不会对操作系统产生恶意操作。但是，由于所分析的代码不一定是最终执行的代码，研究人员可能消耗大量时间分析无用的代码。同时，静态分析对反汇编技术的依赖也使其具有严重的局限性。黑客可使用各种混淆技术阻碍反汇编分析，例如加壳、加密、压缩、变形等保护技术，其完整代码只有在实际运行中才释放，导致静态分析难以得到正确结果。为了解决混淆技术问题，研究人员进行了一系列的尝试，例如 Christodorescu 等人[12]针对代码重排、加壳、垃圾代码插入 3 种混淆方式提出了相应的解决方案，并试图将恶意代码恢复为混淆前的代码，但未能解决寄存器重分配、等价替换等其他混淆技术的干扰问题。动态分析则是基于行为特征的分析技术。在恶意代码执行过程中进行分析，所分析的代码就是实际执行的代码，而且分析过程可以动态监控并记录程序执行时系统调用等行为信息，可以克服静态分析的局限性。但动态分析一次执行过程只能获取单一路径行为，而一些恶意代码存在多条执行路径，甚至执行路径是随机出现的。而且，当前的动态分析技术多是采用行为序列的曼哈顿距离或加权曼哈顿距离进行检测，恶意代码可以使用系统调用重排和加入垃圾调用等方式增大行为之间的距离，进而绕过以行为序列为特征的检测方案。最初的动态分析借助调试工具进行，依赖于安全人员的研究经验，其断点等操作易被恶意代码察觉并采取反制手段。此后，研究人员开发了一系列的动态分析工具辅助人工分析，比较有代表性的是使用虚拟机[13]和硬件模拟器[14,15]的方法。文献[16]指出，对于大规模数据集，动态分析方法时间复杂度过高。通常，动态分析流程为解分组、执行程序逻辑，通过全路径探索罗列调用序列、运行结束和资源释放。

2.2 二进制可视化方法

目前，许多工具都能够可视化和操作二进制数据，例如常用的文本编辑器和二进制编辑器。在可视化软件程序方面也存在相关的研究。在文献[17]中，Yoo 使用自组织映射方法去检测和可视化恶意代码程序。在文献[18]中，Quist 和 Liebrock 为恶意代码逆向工程开发了一套可视化框架，通过节点和链接的可视化鉴别恶意代码的功能区域和混淆区域。在文献[19]中，Trinius 等人使用树形拓扑图表示恶意代码的操作分布，使用线程图表示操作序列。在文献[20]中，Goodall 等人为了辅助开发者更好地理解代码，开发了可视化的分析环境。在该环境中，软件内部的漏洞能够被可视化。Conti 等人在文献[21]中将原始二进制数据段，例如文本、C++数据结构、图像数据、视频数据，可视化为图像。在文献[22]中，Conti 等人基于统计特性将可执行文件自动分类为不同的二进制段，例如代码段、数据段、初始化段等。但是，恶意代码可视化方面研究相对较少，在文献[4]中，Nataraj 等人首次提出将恶意代码可执行文件映射为灰度图像，使用 GIST 算法抽取图像特征，并基于此特征使用 K 最近邻算法进行分类。在文献[23]中，Kancherla 等人将恶意代码可执行文件映射为字节图，抽取并使用 SVM 分类器对规模为 2.5 万的恶意代码语料库进行分类。

3 技术路线

本文的主要研究内容是基于纹理分块的恶意代码纹理特征提取及变种检测方法，包括恶意代码映射、纹理分块和指纹提取，建立恶意代码纹理指纹索引结构、恶意代码近似相似性检索和整体流程算法设计。为了实现纹理指纹提取和变种检测，首先要对恶意代码二进制程序映射为无压缩的灰阶图像，然后基于段自增长的纹理分割算法对图片进行分块，使用灰阶共生矩阵算法提取各个分块的纹理特征（例如能量、熵、惯性矩等），然后使用高斯归一化算法将这些纹理特征进行归一化处理，归一化结果作为恶意代码的纹理指纹；然后，根据获取样本的纹理指纹，使用 M-Trees 算法建立纹理指纹树形索引结构。在恶意代码检测阶段，使用加权综合多分段纹理指纹相似性匹配方法检测未知恶意代码或恶意代码变种。在分析恶意代码检测的研究发展基础上，本文提出的基于纹理指纹的恶意代码变种检测方法主要分为 2 个阶段，恶意代码纹理检测库建立阶段和恶意代码变种检测阶段。该方法的核心主要由段自增长的纹理分割算法、恶意代码纹理指纹提取算法和加权综合多分段纹理指纹相似性匹配方法组成，辅助方法还包括 M-Tree 算法、Top-k 算法等，辅助方法并不作为本文研究重点。

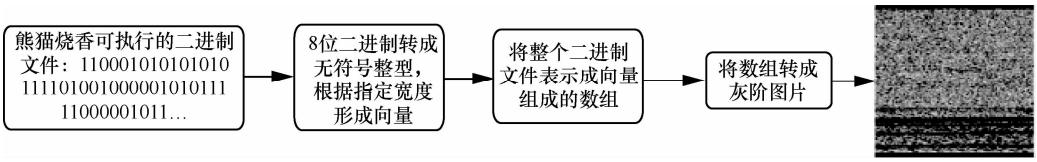


图 1　熊猫烧香恶意代码可视化流程

3.1　B2M 算法描述

对于给定的恶意代码可执行文件，即二进制文件，读取 8 bit 为一个无符号的整型（范围为 0~255），固定的行宽为一个向量，整个文件最后生成一个二维数组。此数组中每个元素的范围都是取值为[0，255] (0 表示黑色，255 表示白色)，将此数组可视化为一个灰阶图像，图像的宽度和高度取决于文件大小，例如宽度选择 PE 文件段宽度（512 byte）的一半为 256 byte，高度为文件大小与 256 的比值。该算法称之为 B2M 算法，映射后的灰度纹理图片将被存储为无压缩的 PNG 图片，图 1 所示为使用 B2M 算法映射恶意代码熊猫烧香到纹理图片的可视化流程。

3.2　建立纹理特征空间

3.2.1　灰度共生矩阵纹理特征

使用暴力匹配方式搜索包含百万级数量灰度图像的大规模纹理图像数据集是不现实的。当前，大规模图像检索方法通常使用图像的纹理特征描述符表示其纹理特征。纹理是对图像象素灰度级在空间上分布模式的描述，是图像的区域特征，反映图像中物品的质地，如粗糙度、光滑性、颗粒度、随机性和规范性等。通过纹理分析，可以获得图像中物品的重要描述信息，是图像分块、特征提取和学习识别的重要手段。而图像分块中纹理的边界可以通过计算纹理特征是否发生显著改变来决定。灰度共生矩阵（GLCM, gray level co-occurrence matrix）是用 2 个位置的象素的联合概率密度来定义的，它不仅反映亮度的分布特性，也反映具有同样亮度或接近亮度的象素之间的位置分布特性，是有关图像亮度变化的二阶统计特征。它是定义一组纹理特征的基础。一幅图像的灰度共生矩阵能反映出图像灰度关于方向、相邻间隔、变化幅度的综合信息，它是分析图像的局部模式和它们排列规则的基础。纹理分析中计算图像的纹理特征量，并不直接使用灰度共生矩阵，而是在灰度共生矩阵的基础上再抽取纹理特征量，称为二次统计量。Haralick 等人[24]由灰度共生矩阵提取了 14 种特征。为了降低维度，同时保持数据集中的对方差贡献最大的特征，本文使用 PCA 算法[25]对 14 种特征进行处理，选取了贡献最大的 6 个特征，分别为 Contrast、Homogeneity、Correlation、Dissimilarity、ASM、Entropy，统称为 GLCM-6。图 2 为极虎病毒 PE 二进制文件的纹理，其 GLCM 的 6 个特征分别为 10.734 7、0.362 59、0.137 31、2.508 6、0.023 11、0.152 02。

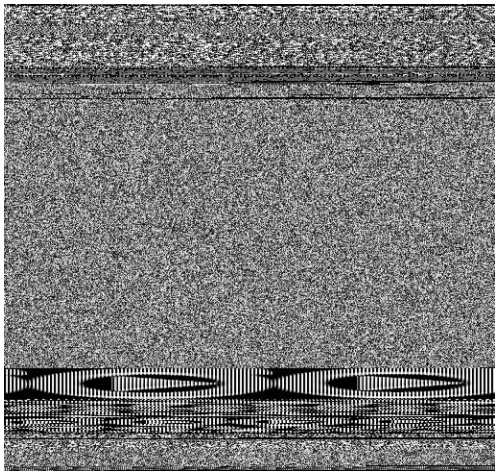


图 2　极虎病毒纹理

3.2.2　特征规范化

对于单一纹理特征的数据集，数据集的规模 N 是常量，纹理特征值相等的越少表明这个数据集内部的不同状态越多，即数据集的复杂度越高，用信息熵公式 $H=-\sum_{i=1}^{k} p_i \log p_i$ 表示一个数据集的不确定性。数据集的整体复杂性可以定义为 $C=NH$。根据信息熵的定义，不确定性与具体纹理特征值的大小和单位无关。当 M 个纹理特征维度具有 $n_1,n_2,\cdots,n_m$ 个不同的取值时，使用 KNN 查询时返回的数据需要反映这种组合复杂度的不确定性。而且，独立纹理特征索引返回的数据量与组合后纹理特征向量的整体维度之间存在反比关系，由于维度越小的索引在整体中作用相对较小，因此索引需返回的数量就越多。使用 KNN 查询时每个索引需返回的查询结果定义为

$$k(H_1+H_2+\cdots+H_m)=k\log(n_1,n_2,\cdots,n_m)\sum_{i=1}^{M}\sum_{j=1}^{n}D_j/D_i$$

其中，D_i 为该纹理索引所对应的维度。在所有维度查询结果返回后，根据用户给出的权重 W_i 加权来比较返回最邻近节点的数据。相似度定义为 $S(N,P)=\sum_{i=1}^{n}D(N_i,P_i)W_i/\sum_{i=1}^{n}W_i$，其中，距离函数 D 可以采用欧拉距离、海明距离等。

以恶意代码纹理图像搜索为例，对于不同的纹理特征提取算法，其数据格式与取值范围存在很大的不同。若仅仅简单地把某张纹理图片的所有特征向量连接起来，并建立索引结构，则会导致某些特征的影响被放大，而某些特征的影响被忽略的情形。空间欧氏距离的大小往往依赖于数值分布范围较大的数据特征，当数据中某几维属性的取值范围相比其他属性过大时，计算出的欧式距离通常仅反映了取值较大的特征维度上的差异性。但在现实中，数值取值较小的属性很可能对分类具有至关重要的作用，因此对于基于欧式距离的相似度函数，数据的规范化至关重要。因此，为了避免计算相似性距离时忽视取值较小的特征，对纹理特征进行规范化的算法如下：对于 n 维特征向量 $\boldsymbol{v}^{(n)}=(v_1,v_2,\cdots,v_n)$，使用高斯公式，计算特征向量集合的均值 μ_n 和方差 σ_n，然后将 $\boldsymbol{v}^{(n)}$ 归一化至[−1,1]区间，从而得到归一化的 $\boldsymbol{v}^{(N)}$：$\boldsymbol{v}^{(N)}=\dfrac{v_i-\mu_i}{3\sigma_i}$，$i$=1,2,⋯,$n$。其中，上标 N 表示归一化。归一化后，各个分量均转变成具有 $N(0,1)$分布的 $\boldsymbol{v}^{(N)}$。使用 $2\sigma_i$ 进行归一化，保证了 $\boldsymbol{v}^{(N)}$ 的值落在[−1,1]区间上的概率接近 100%，对于离群点，小于−1 则置为−1，大于 1 则置为 1。在 MVDS-TF 检测系统中，n 为 6，即使用了 6 个特征量近似表示子域的纹理特征。

3.3 索引结构

GLCM-6 为 6 维特征向量，为有效的最近邻查询，本文分别引入了 M-Tree 数据结构和 LSH 数据结构进行比较，通过详细的实验验证和比较，选择效率总体最优的索引结构用于 MVDS-TF 检测系统中。

M-Tree[26]是类似 R-Tree 和 B-Tree 的树形数据结构。它是用一个度量函数构建的，度量函数需要确定满足自反性、非负性、对称性以及三角不等式，使用 KNN 查询。为了降低高代价的距离计算，M-Tree 索引结构将大多数距离预计算好并存储在索引结构中。这样就可以避免很多距离的动态计算。对于一个规模为 M，维度为 N 的样本集合，M-Tree 的查询时间复杂度为 $O[N\ \log(M)]$，暴力搜索的时间复杂度为 $O[MN]$。

LSH 算法首先由 Indyk 等人[27]提出，用来解决空间中高维向量的近邻相似性检索问题，能够证明其对数据规模 n 具有线性时间复杂度。它的关键思想是使用若干散列函数，确保空间距离度量近的点比距离度量远的数据点冲突概率大，当要检索时，仅检索与待查数据点 q 冲突的点，从而很大程度上减少了距离计算，加快检索时间。在文献[28]中，作者提出一个以二进制海明距离为度量方式的位置敏感散列函数，已经在很多领域中应用，但它有一个明显的缺点，通常距离度量函数，都是欧拉距离，要应用此算法，必须将欧拉距离转换为二进制海明距离，这将增加算法的检索时间和复杂性，为了提高算法的效率和通用性，在文献[29]中，作者提出了基于 P-Stable 分布的位置敏感散列算法，该算法可以直接处理二次欧拉距离，并解决了(R, c)−NN 问题，其时间复杂度为 $O(n^c\ \log\ n)$，空间复杂度为 $O(n^{c+1})$。另外，LSH 算法处理高维稀疏数据效果较好，特别是当高维向量中非零元素规模一定时，算法的检索时间不变，这个特性是独一无二的。因此，使用 LSH 算法处理高维稀疏数据比线性扫描具有更大优势。

3.4 恶意代码变种检测

恶意代码家族是指有具有明显特征的恶意代码种类，是由很多拥有共同特性的恶意代码个体组成，共同特性通常包括相同的代码、图案、应用特征及相似的行为方式。恶意代码家族中的个体成员之间差异较小，而且它们的基因结构也比较相近，犹如物种在进化过程中基因变化一样，如图 3 所示，恶意代码家族 Worm.Win32.WBNA（venus_ 1ef51f0c0ea5094b7424ae72329514eb、venus_ 777b2c29dc6b0c0ad1cec234876a97df 、 venus_1701f09f7348b0a609b8819b076bb4df）中的 3 个成员，查看其 PE 文件纹理变化情况。图 3 从上到下分为 3 层，每层分为 3 列。第 1 层从左到右为 3 个恶意代码的纹理图像，第 2 层从左到右为第 1 层 3 个纹理图像的差异（与第 1 层的第 1 幅纹理图像比较），第 3 层为恶意代码标识，命名规则为“company+md5”。由图 3 可见，同属于一个恶意代码家族的恶意代码在 PE 文件内容和纹理上（可视化）具有很大程度的相似

性，因此可以通过恶意代码整体内容纹理上的相似性来判断恶意代码是否是某个家族的变种。但是，这个朴素方法存在一个较大的弊端，对于代码重排、垃圾代码插入等混淆方法无能为力，为了解决此类问题，本文提出基于段自增长的纹理分割算法，将恶意代码纹理文件进行分割，使用分割后的代码块进行特征匹配。

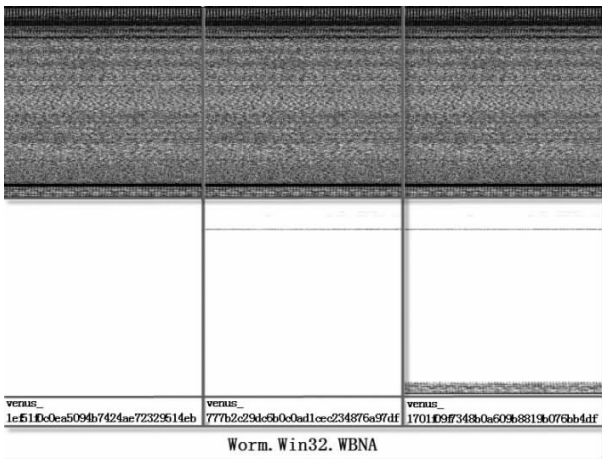


图 3　Worm.Win32.WBNA 3 个家族成员内容纹理差异实例

3.5　段自增长的纹理分割算法

3.5.1　恶意代码纹理分块形式化描述

由于恶意代码二进制文件是以 PE 文件格式组织的，文件内容被分割为不同的区段，每一区段中可能包含代码或数据。各区段按页边界对齐，区段没有大小限制，是一个连续结构。因此，为了降低纹理分割的时间复杂度，参照图像分割领域的区域增长算法[30]，结合恶意代码纹理分割的实际应用，本文提出了基于段自增长的纹理分割算法，算法的形式化描述如下。

1) R_i 是一个连通域，$\bigcup_{i=1}^{n} R_i = R, i = 1,2,\cdots,n$。该断言表示分段完成后，每个段必须都在相应的域中，并且子域中的点必须是连通的。即具有不同恶意代码纹理特征的块将分到不同的子域中，相同域中的块具有相似的纹理特征。

2) $R_i \cap R_j = \varnothing, i, j = 1,2,\cdots,n$。该断言表示域之间是不相交的，即恶意代码纹理子域之间具有不同的特征。

3) $P(R_i) = \mathrm{TRUE}, i = 1,2,\cdots,n$。该断言表示在同一子域中的段具有相似的纹理特征。

4) 对于邻域 R_i 和 R_j，$P(R_i \cup R_j) = \mathrm{FALSE}$。该断言表示在不同子域中的段具有不同的或差异较大的纹理特征。其中，$P(R_i)$ 是对所有在集合 R_i 中元素的形式化表达式，$\varnothing$ 为空集合。

3.5.2　恶意代码智能纹理分割算法

通常，组成恶意代码二进制 PE 文件的段长度（数据块或扇区）为 512 byte，在映射图像过程中，将图像宽度设定为 256 byte，即每两行纹理段表示一个单位段。基于段自增长的纹理分割算法对恶意代码极虎病毒纹理文件分块结果如图 4 所示，具体的分块步骤如下。

step1　把所有的段都标以“未完成”，即都没有被合并，然后按照段顺序扫描整个恶意代码纹理文件。

step2　对恶意代码纹理文件段按顺序扫描，找到第 1 个还没有归属的纹理段，设该段为(x_0)。

step3　以(x_0)为分析对象，判断(x_0)对象是否满足退化准则，若满足则标记该段为“完成”。返回，继续分析下一行纹理段。

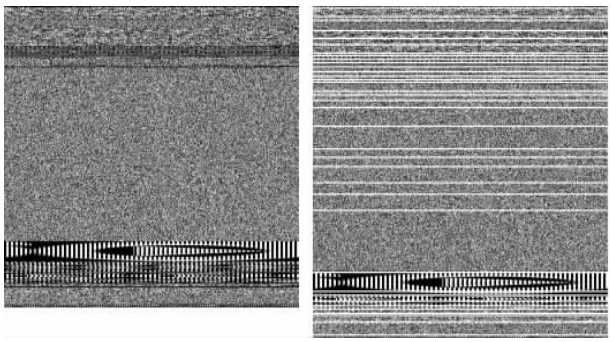
step4　否则，考虑(x_0)的下邻域纹理段(x_1)，如果(x_1)满足生长准则，将(x_1)与(x_0)合并(在同一区域内)，同时标记该段为“完成”。返回，继续分析下一行纹理段。

step5　重复读取恶意代码纹理文件的纹理段，进行 step2 到 step4 的操作。

step6　当处理完恶意代码纹理文件最后一行时，停止生长，返回图像纹理的分块结果。

自生长准则：若当前纹理段与上邻域段纹理特征距离小于给定的阈值，则将当前纹理段与上邻域合并形成新的子域，否则停止生长，当前纹理段作为新域起始段。

退化准则：若纹理段是常量组成的纹理，即计算其灰度共生矩阵的对比度为 0，子域停止生长，删除该纹理段。



(a) 原始样本　　(b) 分块后的样本(白线条表示分割线)

图 4　极虎病毒样本分块

3.5.3 恶意代码纹理块策略生成

根据对恶意代码语料库分块的统计和分析结果，对恶意代码纹理块策略定义如下。

1) 块权重：将恶意代码纹理文件进行智能分块，将第 k 块 B_k 中段数 C_n(即行数，每行为 256 byte)与文件的总段数的比值记为块权重 w_k。

2) 干扰块：干扰块定义为干扰检测系统正确判断，提高误报率的纹理块。干扰块的来源主要有 2 个方面，一是 Windows 操作系统的系统文件的分块，另一个是恶意代码纹理块的段数过小（例如 $C_n \leqslant 3$）。在对恶意代码语料库中的恶意代码纹理文件进行块可信度统计和建立索引结构时，要删除此类干扰块。例如，对 Windows 操作系统的系统文件（exe 文件、dll 文件等）进行分块，然后使用这些块文件的 md5 码进行过滤。

3) 块可信度：对恶意代码语料库中的恶意代码纹理文件进行分块，然后使用 md5 算法进行块统计，设块出现频率为 C_f，通过对语料库的统计和人工分析结果，设定三级块可信度阈值，分别为高可信度频率 Cr_{H}、中可信度频率 Cr_{M}，一般可信度频率 Cr_{G}。例如，对于 $C_f \geqslant 10$ 的纹理块设定为高可信度块，对于 $5 \leqslant C_f \leqslant 10$ 的纹理块设定为中度可信度块，对于 $C_f \leqslant 3$ 的纹理块设定为一般可信度块。

4) 匹配：对于待检测恶意代码纹理块 C_k，通过索引结构返回若干结果，记为候选数据集 C_s。计算 C_k 与 C_s 集合中其他块的欧式距离，记为 D_k，若 D_k 小于某阈值，则定义为 C_k 与 C_s 集合中的当前块匹配。例如本文使用 $D_k \leqslant 0.05$，记为匹配。

为了能够对 MVDS-TF 系统的检测返回结果进行评价，根据以上定义，对于结果的评价策略如下。

1) 块可信度评分：对于返回的候选结果块，高可信度频率 Cr_{H} 记为 5 分、中可信度频率 Cr_{M} 记为 3 分，一般可信度频率 Cr_{G} 记为 1 分。若待检测恶意代码块可信度总评分 $C_{\mathrm{T}} \geqslant 10$ 记为确认恶意代码，$5 \leqslant C_{\mathrm{T}} \leqslant 10$ 记为疑似，$1 \leqslant C_{\mathrm{T}} \leqslant 5$ 记为可疑，$C_{\mathrm{T}} \leqslant 1$ 记为未知。

2) 块权重距离：对于恶意代码纹理文件与恶意代码语料库中的恶意代码文件距离定义如下，$D = \sum_{i=0}^{k}(1-w_i)D_i$，其中，$w_i$ 为块权重，D_i 为匹配的最小距离，不匹配则为 0。

3) 投票策略：对于待检测恶意代码纹理块 B_k，在索引结构中与其匹配的数据块个数为 B_m 个，每个数据块对应了 M_p 个恶意代码，即与纹理块 B_k 相对应的恶意代码有 $B_m \times M_p$ 个，记为集合 S_k。恶意代码的家族 $M_{\mathrm{F}} = \bigcap_{i=0}^{k} S_i$，通过候选恶意代码集投票来决定待检测恶意代码的家族。

4 实验数据及结果分析

4.1 实验环境和测试样本集

基于纹理指纹的恶意代码变种检测系统（MVDS-TF, malicious code variants detection based on texture fingerprint），运行的硬件环境为 IBM 服务器（2 颗英特尔四核至强 E5420 2.5 GHz/EM64T，12 MB L2 缓存），配置 32 G PC2-5300 CL5 ECC DDR2 667 MHz 内存、3 块 146 G 硬盘；操作系统使用 CentOS 6.4 64 位，后台数据库使用 MongoDB 2.2.6。恶意代码映射过程、分块、特征抽取、检索和过滤过程使用 Python 语言，相关包为 Anaconda-1.8.0-Linux-x86_64，包括了工程所有涉及到的包。其中，MongoDB 存储了恶意代码样本的相关信息，例如 MD5 值、文件大小、家族标注、恶意代码 PE 块相关信息等。

实验过程中，本文使用 Venustech 提供的恶意代码语料库（规模为 20 490 个恶意代码，venus-2M，分块后的块数为 5×105）进行分块生成索引结构。测试样本集分成 Hacktool、EvilJs、Packed、Trojan-Dropper、Worm 和 Backdoor 类型，基本信息如表 1 所示。

表 1 恶意代码测试样本集说明

恶意代码测试样本集	规模	总大小/MB	大小范围
Hacktool	126	98.5	4.1 kB~22.0 MB
EvilJs	168	7.5	1.1 kB~2.0 MB
Packed	326	211.7	39 kB~9.8 MB
Trojan-Dropper	339	423.1	11 kB~9.9 MB
Worm	452	256.5	10.9 kB~9.9 MB
Backdoor	537	505.4	2.4 kB~22.2 MB

4.2 MVDS-TF 检测过程

在本文的实验中，根据恶意代码样本与其变种在 PE 文件内容上共性的特点，本文提出基于内容指纹的恶意代码变种检测方法，检测过程如图 5 所示，具体实验过程如下。

1) 根据恶意代码语料库中的恶意代码样本建立索引结构（M-Trees 或 LSH），索引结构建立过程与下面检测过程相似。

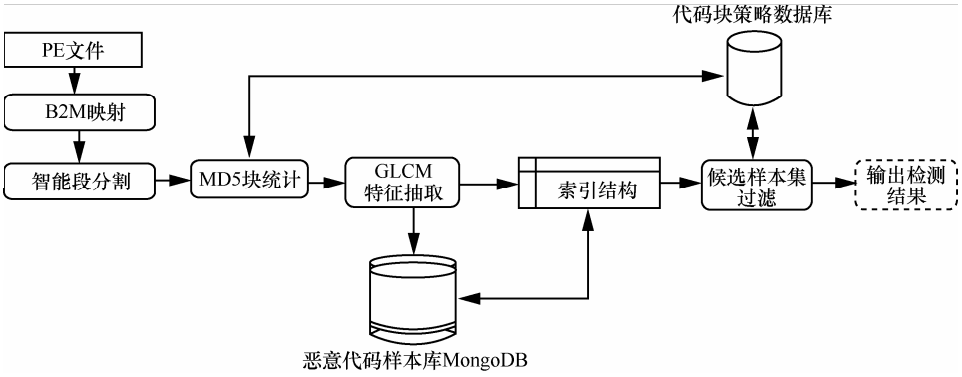


图 5　MVDS-TF 检测流程

2) 使用 B2M 算法映射待检测恶意代码 PE 文件为 PNG 格式的灰度图像。

3) 使用段自增长的纹理分割算法，对该图像进行智能分块。

4) 利用灰度共生矩阵提取灰度图像各个分块的纹理特征向量 GLCM-6，也称之为内容指纹向量。

5) 利用高斯归一化算法对各分块的 GLCM-6 特征向量归一化，特征向量取值均在[0,1]，以保证分量值在距离计算时具有相同的地位。

6) 使用步骤 1)建立的索引结构，获得该恶意代码各个分块内容指纹的最近邻集合。根据代码块策略处理各块的最近邻集合，输出检测结果。

4.3　实验结果分析

为了验证 M-Trees 和 LSH 这 2 种索引结构的时间性能以及特征维度对它们的影响，使用开源的 python LSHash 和 Sklearn 的 M-Trees API 索引 venus-2M 恶意代码语料库，并使用 Hacktool 测试集进行性能测试，测试结果如图 6 和图 7 所示。LSHash(10,6,1)各个参数分别表示使用 10 位二进制数表示散列桶，纹理特征数据集维度为 6，即 GLCM-6，散列表的个数为 1。如图 6 所示，随着散列表的个数的提高，则查询过程需要的时间也随之提升，同时召回率(recall)会提高，而由此产生的误报率也会变高。因此，根据语料库的规模和误报率适当选择散列表数，本文选择散列表数为 3。为了比较 M-Trees 和 LSH 的时间性能以及特征维度对它们的影响，分别使用 GLCM-6 和 GIST[4,5]特征对恶意代码纹理进行描述，特征维度为 6 和 320，比较结果如图 7 所示，随着维度的提高，LSHash 的性能呈现退化趋势，而 M-Trees 的性能变化不大，而且 M-Trees 的时间性能优于 LSHash。因此，MVDS-TF 恶意代码检测系统选择 M-Trees 作为索引结构。

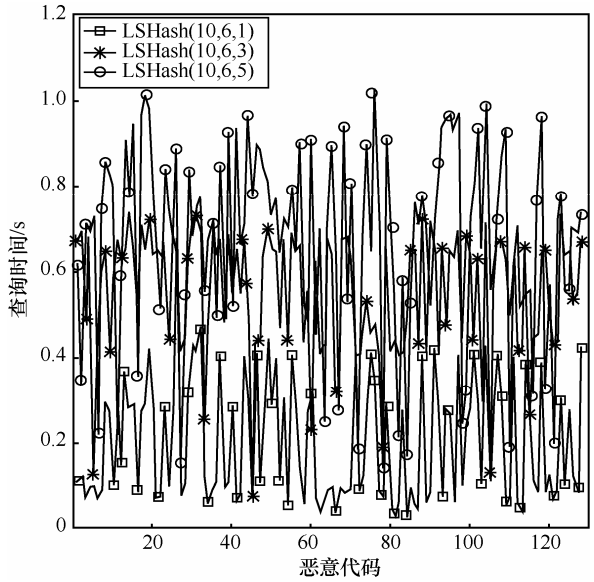


图 6　LSH 索引结构时间性能与散列函数个数关系（d=5, $size$=5×10^6）

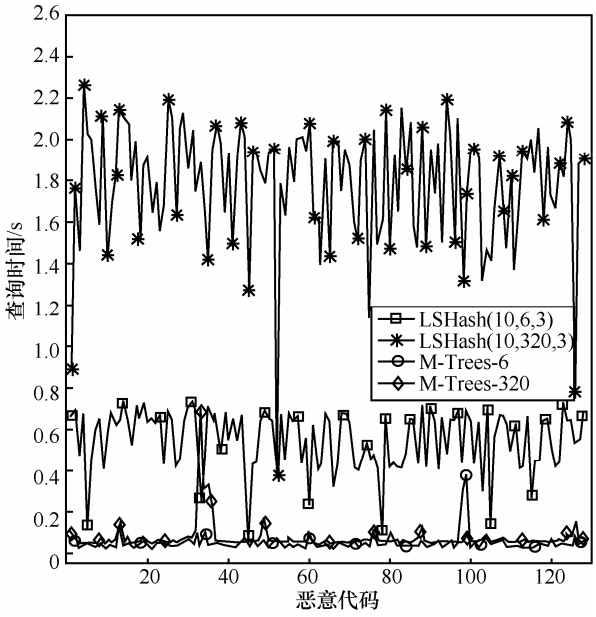


图 7　M-Tree 索引结构与 LSH 索引结构时间性能比较（$size$=5×10^5）

MVDS-TF 检测过程大致可以分为 5 个阶段，分别为投影（map）、分块（block）、特征抽取（feature）、检索（search）和过滤（filter）。对于每个待检测的恶意样本，MVDS-TF 检测时间主要是用在这 5 个阶段。为了评测各阶段所用时间的情况，选择 Hacktool、EvilJs、Packed、Trojan-Dropper、Worm 和 Backdoor 6 种测试数据集，测试结果如图 8 和图 9 所示。图 8 表示 6 个数据集的规模（个数）和恶意代码总大小（MB），图 9 表示这 6 个数据集在 MVDS-TF 检测过程 5 个阶段所耗费的时间代价。总体而言，对于不同的数据集，检测 5 个阶段所花的时间各不相同。本质上，MVDS-TF 检测各阶段所花的时间与数据集的总大小有关，图 8 中数据集大小曲线和图 9 中各阶段的曲线趋势大致相同，即对于单个待检测恶意代码样本来说，MVDS-TF 检测所花费的时间仅仅与样本大小有关，样本越大，检测花费的时间代价越大。

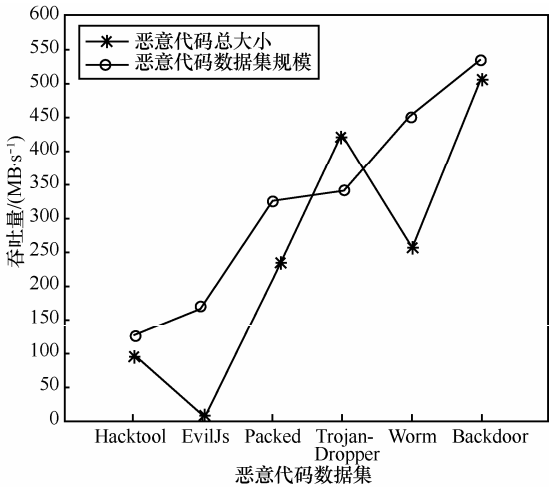


图 8 系统吞吐量与恶意代码数据集规模和大小关系

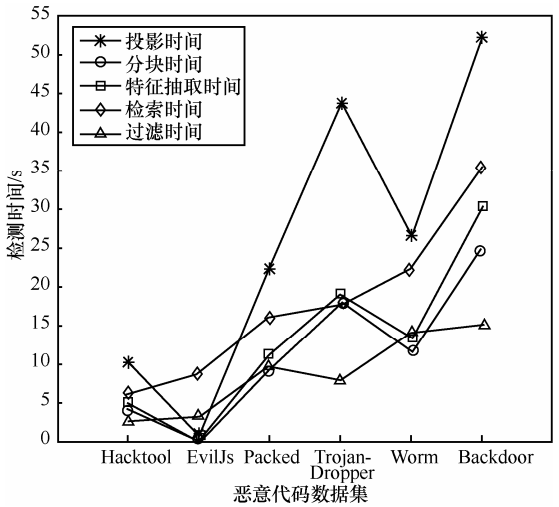


图 9 检测各阶段耗费时间与恶意代码数据集关系

从 Hacktool、EvilJs、Packed、Trojan-Dropper、Worm 和 Backdoor 6 种测试数据集中每个数据集随机取 50 个恶意代码，然后使用 venus-2M 恶意代码语料集建立索引时，排除这 300 个恶意代码，使用全内容检测（使用整个恶意代码纹理文件建立索引）和智能分块（利用恶意代码分块后的纹理文件建立索引）检测分别进行实验，得到的实验结果如表 2 所示。从表 2 中可以看出，本文提出的检测方案能够很好地提高恶意代码检测的准确率，并且通过使用恶意代码纹理块策略中的投票策略，MVDS-TF 检测系统能够正确识别出恶意代码的类型。

表 2 恶意代码变种检测准确率

恶意代码测试样本集	规模	未分块/%	分块/%
Hacktool	126	71.72	79.20
EvilJs	168	77.10	81.18
Packed	326	51.22	60.10
Trojan-Dropper	339	74.00	80.18
Worm	452	76.49	85.47
Backdoor	537	79.81	85.77

5 结束语

基于图像纹理处理方法，本文提出一种基于内容指纹的恶意代码变种检测方法，从恶意代码二进制文件的块内容相似性（映射为纹理图后，可理解为视觉相似性）检测恶意代码的变种。通过对种类恶意代码样本进行实验测试，验证了本文提出的基于内容纹理指纹的恶意代码变种检测方法的有效性。本文提出的特征提取及检测方法提高了对恶意代码变种的检测能力，对正常程序与恶意代码有较好的识别度，且在很大程度上降低了特征数据量和维度。在实验中发现，当恶意样本语料库规模较小时，MVDS-TF 检测系统对于加壳、混淆的恶意代码检测效率并不十分理想。总结其原因主要如下。

1) venus-2M 恶意代码语料库过小，很多恶意样本并没有包含在内。

2) 对于恶意代码样本采用不同的加壳技术，会生成不同的纹理，在构建索引结构时，仅对少数几种主流加壳方法进行分析和处理；。

3) 纹理分割算法对于某些种类的恶意代码效果不好。

对于以上问题，拟在扩充语料库、加壳软件分

析和探索更优的纹理分割算法 3 个方面进行改进研究，提高检测精度。此外，随着恶意样本规模逐渐增大，MVDS-TF 检测系统的检测时间将不能满足实时应用要求。因此，结合类似于分布式计算框架 Spark 的技术，分布式检索算法是未来值得研究的方向。

参考文献：

[1] SYMANTEC. Highlights from 2010 internet security threat report [EB/OL].http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=threat_report_16,2011.

[2] SYMANTEC. Highlights from 2012 internet security threat report [EB/OL]. http://www.symantec.com/security_response/publications/threatreport.jsp,2013.

[3] LI Y, ZUO Z H. An overview of object-code obfuscation technologies[J]. Journal of Computer Technology and Development, 2007, 17(4):125-127.

[4] NATARAJ L, KARTHIKEYAN S, JACOB G, *et al*. Malware images: visualization and automatic classification[A]. Proceedings of VizSec[C]. Pittsburgh, USA, 2011.

[5] NATARAJ L, YEGNESWARAN V, PORRAS P, *et al*. A comparative assessment of malware classification using binary texture analysis and dynamic analysis[A]. Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence[C]. Chicago, USA, 2011.21-30.

[6] 王蕊, 冯登国, 杨轶等. 基于语义的恶意代码行为特征提取及检测方法[J]. 软件学报, 2012,23(2):378-393.
WANG R, FENG D G, YANG Y, *et al*. Semantics-based malware behavior signature extraction and detection method[J]. Journal of Software, 2012, 23(2):378-393.

[7] COGSWELL B, RUSSINOVICH M. Rootkit revealer[EB/OL]. http://www.microsoft.com/technet/sysinternals/Utilities/RootkitRevealer.mspx, 2006.

[8] KIRDA E, KRUEGEL C, BANKS G, *et al*. Behavior-based spyware detection[A]. Proceedings of the 15th USENIX Security Symposium[C]. Canada, 2006.273-288.

[9] CHRISTODORESCU M, JHA S, SESHIA S A, *et al*. Semantics-aware malware detection[A]. Proc of the 2005 IEEE Symposium on Security and Privacy[C]. California, USA, 2005.32-46.

[10] KINDER J, KATZENBEISSER S, SCHALLHART C, *et al*. Detecting malicious code by model checking[J].Detection of Intrusions and Malware, and Vulnerability Assessment, 2005,3548:174-187.

[11] SATHYANARAYAN V S, KOHLI P, BRUHADESHWAR B. Signature generation and detection of malware families[A]. Proc of the 13th Austalasian Conf on Information Security and Privacy[C]. Wollongong, Australia, 2008.336-349.

[12] CHRISTODORESCU M, KINDER J, JHA S, *et al*. Malware Normalization[R]. Technical Report 1539, Madison: University of Wisconsin, 2005.

[13] WILLEMS C, HOLZ T, FREILING F. Toward automated dynamic malware analysis using CWSandbox[J]. IEEE Security and Privacy, 2007, 5(2):32-39.

[14] BAYER U, KRUEGEL C, KIRDA E. TTANALYZE. A tool for analyzing malware[A]. 15th European Institute for Computer Antivirus Research (EICAR 2006)[C]. Hamburg, Germany, 2006.180-192.

[15] BELLARD F. QEMU, A fast and portable dynamic translator[A]. USENIX Annual Technical Conference, FREENIX Track[C]. California, USA, 2005.41-46.

[16] LI P, LIU L, GAO D, *et al*. On challenges in evaluating malware clustering[A]. Recent Advances in Intrusion Detection[C]. Ottawa, Canada, 2010.238-255.

[17] YOO I. Visualizing windows executable viruses using self-organizing maps[A]. International Workshop on Visualization for Cyber Security (VizSec)[C]. Washington DC, USA, 2004.82-89.

[18] QUIST D A, LIEBROCK L M. Visualizing compiled executables for malware analysis[A]. International Workshop on Visualization for Cyber Security (VizSec)[C]. Atlantic City, USA, 2009.27-32.

[19] TRINIUS P, HOLZ T, GOBEL J, *et al*. Visual analysis of malware behavior using treemaps and thread graphs[A]. International Workshop on Visualization for Cyber Security (VizSec)[C]. Atlantic City, USA, 2009.33-38.

[20] GOODALL J H, RANDWAN H, HALSETH L. Visual analysis of code security[A]. International Workshop on Visualization for Cyber Security (VizSec)[C]. Ottawa, Canada, 2010.46-51.

[21] CONTI G, BRATUS S, SANGSTER B, *et al*. Automated mapping of large binary objects using primitive fragment type classification[J]. Digital Forensics Research Conference (DFRWS), 2010, 7:3-12.

[22] CONTI G, BRATUS, S. Voyage of the reverser: a visual study of binary species[A]. Black Hat[C]. USA. 2010.

[23] KANCHERLA K, MUKKAMALA S. Image visualization based malware detection[A]. Computational Intelligence in Cyber Security (CICS)[C]. Singapore, 2013.40-44.

[24] HARALICK R M, SHANMUGAM K, DINSTEIN I H. Textural features for image classification[J]. IEEE Transactions on Systems, Man and Cybernetics, 1973,(6): 610-621.

[25] JOLLIFFE I. Principal Component Analysis[M]. USA: John Wiley & Sons, Ltd, 2005.

[26] PAOLO C, MARCO P, PAVEL Z. M-tree: an efficient access method for similarity search in metric spaces[A]. Proceedings of the 23rd International Conference on Very Large Data Bases[C]. San Francisco, USA, 1997.426-435.

[27] INDYK P, MOTWANI R. Approximate nearest neighbors: towards removing the curse of dimensionality[A]. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing[C]. New York, USA, 1998.604-613.

[28] GIONIS A, INDYK P, MOTWANI R. Similarity search in high dimensions via hashing[A]. VLDB'99: Proceedings of the 25th International Conference on Very Large Data Bases[C]. San Francisco, CA, USA, 1999.518-529.