可视化编译的可执行文件以进行恶意代码分析

非官方中文译本 • 安天实验室 译注

文档信息			
论文题目	Visualizing Compiled Executables for Malware Analysis		
论文作者	Daniel A. Quist，Lorie M. Liebrock		
发布单位	新墨西哥理工大学		
原文链接/出处	http://www.net-security.org/dl/articles/dquist-vizsec09.pdf		
论文发布日期	单击此处输入日期。	**译文发布日期**	2014/9/30
论文摘要&关键词	**摘要**：这个论文提出了一个使用执行程序动态分析的方法，来可视化描绘程序的整体工作流。我们使用 Ether 管理程序框架来隐秘监控一个程序。数据将被处理并上传来逆向工程。通过使用这个方法，提取一个运行程序的关键特性所需要的时间将大大减少，提高了效率。 **关键词**：逆向工程；可视化；动态分析		
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。

可视化编译的可执行文件以进行恶意代码分析

Daniel A. Quist
新墨西哥理工大学
洛斯阿拉莫斯国家实验室

Lorie M. Liebrock
新墨西哥理工大学

摘要:逆向工程编译的可执行文件是一件拥有陡峭学习曲线的工作。将汇编语言翻译为一系列表示程序的整体流程的抽象概念使得该任务进一步复杂化。大部分步骤包含找到一个可执行程序的关键领域，并确定它们的整体功能。本文提出了一个动态分析程序执行对的方法,来可视化地表示程序的整体流程。我们使用 Ether 超级管理程序框架来隐秘地监控一个程序。数据将被处理并传输给逆向工程师。通过这个方法,提取一个可执行文件的关键特征所需要的时间将大大减少,提高了效率。初级用户研究表明这个工具对新用户和经验丰富的用户都非常有用。

关键词：逆向工程；可视化；动态分析
索引词：K.6.1[计算管理和信息系统]：工程和人员管理--生命周期；K.7.M[计算职业]：混杂--规范

1. 简介

现今编译的可执行程序都非常大而复杂。此外，可用于逆向工程的语言是汇编的,并且分析程序是一件令人望而生畏的事情。掌握逆向工程的学习曲线是非常陡峭的。一旦掌握了这项技能,这个过程本质上就是一项密集劳动了，且代价昂贵。

逆向工程的首要任务是确定程序的功能。虽然确定代码的目的很有价值，但是识别代码比知道其功能更困难。逆向工程使用很多工具来帮助完成这项工作,包括静态反汇编器、调试器、系统调用追踪工具和脚本工具。追踪程序的执行过程是很困难的,这是因为恶意软件试图规避检测。现代工具必须测量和分析可执行文件而不被发现。

更多情况是，当分析一个指定程序时，最难的事情是找到程序的起始点。所获得的信息可能非常多。经验和技术可以提供帮助。减少理解程序整体框架所花费的时间将大大提高逆向工程的效率。当一个程序由打包工具和其他软件防御工具模糊后,将进一步减缓分析。在这个前提下，我们提出了该框架来促进初始程序理解和去模糊。

工具分为两大类：静态和动态工具。静态分析工具包括反汇编器（例如 IDA 程序），字符串搜索工具，以及特征匹配系统（包括很多反病毒程序）。动态分析工具包括系统调用和操作系统状态追踪，例如 Sysinternals 的 Processmon 工具，和调试器，例如 OllyDbg, SoftICE 和 GDB。利用这些工具，可执行文件的运行时间将被严格控制。

为了帮助逆向工程，我们首先确定下述目标。首先，快速定位压缩程序的初始入口点。这是一个大工程，其复杂度由程序的复杂度决定。第二，帮助理解程序的整体组成部分，重点是程序中的最常执行部分。

本文做出如下贡献:我们的可视化工具可以缩短分析恶意软件和其他可执行文件的过程。通过修改 Ether 框架，我们提供一个隐蔽的监控执行程序的方法。最终，我们整合自己的工具和已经发布的逆向工程工具来加快分析过程。

本文的结构如下。第二章介绍相关的研究。第三章概述了逆向工程工作。第四章提出 VERA（可执行文件逆向和分析的可视化）体系结构。这包括修改 Ether，数据组织和图表布局,以及最终的程序显示系统。第六章应用这个可视化方法来研究 Mebroot 蠕虫。第七章展示了用户研究的结果。最后一章给出了结论和未来工作方向。

2. 之前的相关研究

Dinaburg 等人[5]深入研究过使用硬件虚拟机控制器来监控程序的运行，用来分析恶意软件。我们将在 4.1 节中讨论我们对框架所做的修改。Royal 使用了一个相似的系统，但是该系统具有 Linux 内核虚拟机架构[16]。使用动态检测系统来有效监控程序执行是由 SPiKE 框架提出的[20]。恶意程序的实时调试也已经由 Cifuentes 等人提出,作为一种迅速了解程序执行的方法[4]。Maet 等人已经使用 PIN 系统来追踪恶意代码感染易受攻击服务的过程[13]。分析动态行为的重要性也由 TTangalyze 工具说明了[2][14]。修改传统的虚拟机架构已经被用来帮助分析恶意软件[12,17]。

程序运行的可视化在过去已经被使用,并有良好的效果。所有的分析以提供源代码的程序为中心,或者对没有混淆的代码（例如微软的程序）进行安全分析。Xia 等人监控一个执行程序运行的系统调用,来显示一个进程的感染过程和系统调用流[21]。其他系统,例如那些由 Telesa 和 Voinea 提出的系统证明有效源代码的效果[19]。Bohnet 等人提供一个方法来可视化探索 c 和 c++的源代码[3]。与我们的技术相似，他们强调了提炼一个大型程序（大于 100 万行代码）的基本部分的重要性。这个提

炼过程对于展示一个程序整体流程的相关部分来说非常重要。通过关注编译代码和基本块恶意软件分析，VERA 架构可以从工作中识别出自己。

两个商业产品提供复杂应用程序的静态分析。Zynamics 的 BinNavi 提供一个基于突出程序相关性分析方法的曲线[23]。这个程序通过使用系统调用，突出了程序流和程序结构。HBGary 公司的响应器也关注于软件功能调用，来突出内存访问，变量和系统调用[10]。这两个产品都依赖与调试器接口或者静态分析工具 ,这被证明是不可靠的[6,7,8,]。

3. 逆向工程工作流

逆向工程是一件费时的工作。发现一个程序的功能和目的是非常困难的，并且需要耐心。分析源代码会让人快速疲倦。下面的工作将作为一个逆向未知的程序的高级技术。

在这部分，我们将定义一个简单的方法，我们发现这个方法在逆向工程中非常有用。这不是说这是标准方法，没有标准的存在 ,但是我们发现这个方法是一个高效的逆向工程的方法。

这个过程可以分解为如下步骤 :首先建立一个孤立的环境。接着运行这个程序来获取可识别的输出。使用工具来监控在程序运行中操作系统的变化。第三，加载这个程序到 IDE 程序中，开始逆向工程，如果需要则对二进制文件去模糊。最后，确定和分析相关部分。

虽然可能有很多方法来建立关于程序的功能和的的认知，但是我们发现这个方法是非常有用的。

3.1 建立一个孤立的分析环境

大多数逆向工程都是处理恶意软件或者潜在的恶意代码，这也是本论文的关注点。有一个孤立的环境是非常重要的，用来隔离任何可能发生的恶意行为。习惯做法是使用虚拟系统例如 VMWare 或者 VirtualPC。操作系统通常是 Win XP，代表一般用户的环境。

每个虚拟系统都能够对当前系统状态进行快照。在任何分析执行以前，这个是基线。首先，它提供了一个已知正确的系统，用以与随后的程序执行中的系统状态比较。第二，经常需要恢复到这个配置，来理解一个程序的全部执行过程，并且可以快速恢复到感染以前的状态。一旦运行了快照，可以开始执行和分析程序。

3.2 运行和初始分析

在可控的虚拟环境下运行恶意软件，能够避免感染。这种逆向工程通常被称为动态逆向工程。目标是捕获程序对系统的整体影响，而不关注程序的代码。

这给为我们提供了一个高级别的概览，了解这个软件做了什么。关注影响操作系统的改变，会展示任何修改或者破坏行为的发生。一般的工具例如微软的 Sysinternals 工具监控系统调用运行，文件修改，以及注册表修改。使用工具，例如 Wireshark 可以监控程序的网络活动。

这部分的分析目标是快速确定在散乱的代码中应该找些什么。实际上，在不理解初始行为的情况下，识别反汇编的意思会难很多。

3.3 去模糊

很多新的恶意软件样本都被加壳了。这首先是为了阻止特征检测，调试和逆向工程。在分析前，移除这些代码模糊是一个必要步骤。可以通过任何习惯方法进行移除。典型例子是编码或者加密程序，检测虚拟机，和检测编译器。因为代码是理解程序运行的首要工具，所以要移除各种保护。Guo 等人提供了一个现代程序打包和压缩问题的概述[8]。

解决封装的方法可以依靠静态方法，例如 Guo 提供的方法或者使用虚拟机的动态方法 [5,17,16,12] ,以及缺页辅助调试[15,18]。这些系统依赖能够监控一个运行进程的执行和发去混淆或修改的代码在哪里执行。当这个完成后 ,将执行应用代码的内存转储 ,这样可能发生反汇编。

3.4 反汇编

反汇编代码包括加载到常用的工具中，例如 IDA[11] 或者 OllyDbg[22]。这是代码理解的开始。IDA 提供一个很好的接口来注释和理解代码。绘图方法是原始方法，但是可以帮助高层次解释代码。分析代码可以分为两级：系统调用分析和代码理解。

系统调用分析是寻找相关库调用的过程 ,这些库调用由一部分代码组成。例如如果应用读取一个文件，然后执行一个网络操作，之后关闭一个文件，那么可以合理推断出这个程序执行了一个文件传递。这个是非常高层次的分析技术，提供了优秀的高层次的过程概览。这本质上是基于静态分析，并且通过软件防御系统轻松颠覆。它没有提供应用实际代码执行的信息。

代码理解是发现程序的算法或者底层结构的过程。当发现了有意义的代码，并且还需要更深入理解时，通常执行这个过程。例如，这个分析可以基于系统调用分析推断出的内容。需要被鉴定的特定部分的代码包括加密或认证算法以及任何模糊代码。这个鉴定过程是个耗费人力的过程。

3.5 识别和分析相关部分

在逆向工程中 ,没有固定的方法来发现哪部分是相关

的或者有意义的。虽然已有一些技术，但更多情况是这部分是初学者最困难的步骤。一些技术成功使用考虑关键字符串，考虑关于汇编的相关 API 调用，或者整体检查代码和操作系统间的交互。

查找运行程序中的字符串是大部分人第一步会选择的方法。例如，当逆向程序时，寻找链接向位置网站或网络地址的 URL。这可以用来寻找代码的回首页或者网站交互部分。多数情况下，字符串会采用一个相似的共用格式例如电子邮件地址。一旦可以鉴别这些内容，任何关于它们的内容都可以被发现。大多数情况这将揭露一些程序的邮箱功能。

找到系统调用经常使用的交叉引用是另一步。通过寻找类似文件存取 API，可以追踪任何系统的文件修改。同样，其它相关 API 例如网络流量和修改注册表的 API，都表明系统状态的更改。

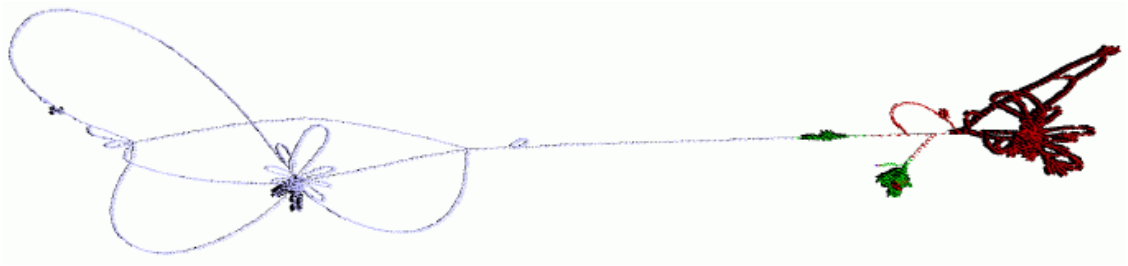

图 1：使用 Mew 打包工具对 Netbull 病毒防护软件的可视化

逆向工程的这个部分是并不精确的基础部分，并且明确我们想改进什么。初学者报告的通常的问题是不知道从代码的哪里开始。他们经常停滞在小细节上，而不是整体上。VERA 追求通过提供一个运行全过程的高层次概览解决这个问题。

4. VERA 架构

我们的工具由三个大部分组成。首先我们改进了基于管理程序的监控框架 Ether 来监控和追踪程序执行，包括内存存取。使用输出的数据，我们构建了一个有向图，展示了可执行文件的所有的基本块（由结点表示）。我们使用开放图显示框架(OGDF)的加权图系统来设计这个图。每个节点的权重是指在分析运行中这部分代码执行的次数。类似的，边的权重是这个控制路径执行的次数。这个数据会通过可视化接口显示。VERA 提供一个导航接口来探测结点。它也链接和联系到 IDA 程序的逆向工具来帮助更多细节分析。

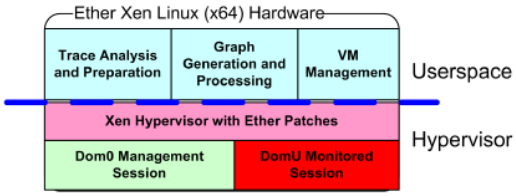

图 2：VERA 和 Ether 架构修改

4.1 用 Ether 进行管理程序监控

Ether 是一系列加入到 Xen 硬件虚拟化框架中的补丁和应用。这个修改让 Xen 能够附属到并且隐蔽监视一个运行的程序。在分析程序时，这给我们一些明显的优点。首先，不会被触发程序验证和保护代码，因此允许程序正常的执行。第二，为了击败传统调试程序和追踪系统的代码模糊无法让 Ether 系统失效。第三，掩藏了虚拟系统的整体状态和结构。程序无法达到发现监控的企图。

我们扩充了 Ether 的一些关键方法，让应用的分析和可视化应用成为可能，参照图 2。使用 Ether 控制程序，我们添加了可以在程序的内读、写和执行记录功能。这给了我们生成程序流程图的必要数据。附加功能确定了合适的时间跳转到执行文件的运行当前状态，这让我们能够避免任何程序内部的打包和模糊。Ether 首先实现的功能包括基于管理程序的解压缩系统。我们已经移动这个逻辑到 dem0 用户空间，使得能够使用更多的传统软件开发工具。最终，我们应用了一个新的外部重构工具来提供更好的 DLL 接口分析。

这个修改的最终结果是一个追踪文件，包括每一个运行状态，内存读取，以及执行指令的反汇编。可执行文件的周期内存快照也被存储用来 IDA 的进一步分析。这个数据之后会处理用来可视化和分析。

4.2 图形分析和布局

VERA 执行以下工作来解析 Ether 生成的追踪文件。首先，解析指令来确定基本块的位置。在我们的可视化程序中，这些基本块将构成结点。基本块之间的执行传递将变为边。执行的次数的计算值决定了边的权重，并且由加粗的边线表示。最后，原始程序的某一特征将通过改变相关结点的颜色来表示。

一旦建立了这个图，我们将使用开放图显示框架（OGDF）来组织和显示这个图。使用这个库让我们能够快速而高效的生成大型图。其它系统例如普遍应用的 GraphViz 无法解决大而复杂的图。我们选择加权对称的

布局选项来组织数据。其它图表工具例如环形布局，我们发现无法及时有效的传达确切的信息。

一旦布局生成，这个布局将被传递到可视化工具 VERA。

5. 可视化和表示：VERA

显示引擎使用一个 2D 视图数据，并将其变为 3D 空间。这为代码提供了更好的缩放和检查特性。这避免了很多 3D 相位的代表数据。虽然 3D 视图通常提供了一个强制的数据视图，但是我们发现在快速初步鉴定上 2D 视图更加有用。为了更容易在工具中集成，图表数据以 OGDF GML 格式描述。这个文件格式包括图表绘图参数例如 X,Y 坐标，以及颜色基数。我们的可视化工具理解和展示了这些数据。

代码的每个顶点代表程序执行的一个基本块。这由所有的两个相邻分值操作间包括的装配操作组成。很多其他程序选择以功能或者方法层的细节来表示数据。我们的决定的背后依据是恶意软件分析。在运行的初始阶段，程序不遵从功能的标准格式。很多代码模糊故意试图发现这个分析工具的功能依赖。

特征的选择如下 :黄色代表执行程序中同时在硬盘和内存中存在的代码。这表示代码在这两个执行状态中是相同的。红色代表执行在一个高信息熵部分。大部分打包工具和模糊工具能够压缩一个可执行文件 ,这样会有均匀分布的数据。程序内部高信息熵的部分表示程序转移到为包装的部分。绿色的部分表示运行到了非法代码部分。如果磁盘中执行程序的执行指令不存在 ,这表示代码动态生成或者自己改变。这些数据区域通常动态分配堆空间，例如通过内存分配返回。浅紫色表示执行部分存在于磁盘，但是不在实时执行。这通常出现在当数据分配在 PE 段头，但是直到执行阶段都没使用。柠檬绿表示执行程序内存中和磁盘执行部分中的指令不同。这是另一种标志指明执行自我修改代码。

一旦数据提交了，视图可以操作来加工信息。缩放，筛选和交互实施方法都类似于谷歌地图接口。通过使用鼠标滚轮完成缩放。在屏幕上点击鼠标左键操作地图。这允许展示执行程序的全部内容，并通过拖拽显示不同部分。通过鼠标移动进行交互。当鼠标停在特定的结点或者基本块时，会显示相关信息。这包括提交数据到 IDA 程序的连接，局部反汇编，引用计数，以及内存修改的区域。右键单击结点可以添加标签，这可以提交到 IDA 程序。

颜色标记会给红绿色盲带去麻烦。为了减少麻烦，提供了多种颜色供选择。可以通过重新编译程序来更新颜色，这是最小代价。

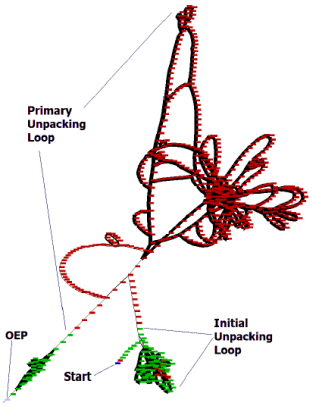

图 3：Mew 未封装环的关闭

图 4：初始未封装环后 Mew 未封装代码的缩放细节视图

通过明确 IDA 和 VERA 间的信息传递，我们能够更好的帮助分析 ,因为用户可以用任一工具对发现的内容做注释，同一注释在两个工具中都可以显示。这让两个工具可以更好的使用，并利用两个工具进行更深入的逆向。

5.1 特征识别

识别程序阶段分为几个独立的工作 :鉴定未封装的代码，鉴定初始化部分以及鉴定程序的主回路。通过很多病毒样本的经验观测值，我们将程序分析分为这些组。很多非恶意程序也显示这些行为。这些概括可以用在鉴定相关行为的大部分情况下。

鉴定程序的未封装环是相对直接的进展。任何一个紧密绑定的循环通常在程序的进入点后很快就可以找到。图 3 展示了 MEW 打包器的解压循环。程序执行的开始是在低端的开始结点。通过这个图表，我们可以看到多重环关联在程序的去混淆中。通过寻找循环出口来鉴定解压循环的结束，并继续到纯色部分。执行程序最长连续部分是浅紫色区域。这个区域的第一个基础模块最有可能是程序的初始入口点。这个声明已被确认，通过比较手动解封装样本和标准解封装方法的观测值。

程序的初始化模块被归为基本块的长链 ,它只有一个入口和出口。在 Figure 1 中，这在中间部分，开始于初始进入节点并结束于右边三个分支发生。执行程序的这些部分典型的处理了分配存储器 ,打开文件和将使用的资源，以及接入网络资源。重要的是注意初始部分的大多数结点

都不局限于这一区域，在后边的部分也可以找到。找到初始化的综合区域是缩小关注区域的重要一步。

最终的关心区域由主执行环组成。在图 1 的最左边区域的中央。暗色的边表示多次运行循环。通过提炼我们对这部分的分析，我们可以找到程序主要的后门部分。这个代码自我激活，执行初始回调，并等待后续的连接。

6. 分析的应用：MEBROOT

为了展示在真实数据下VERA的实际应用，我们使用VERA分析木马Mebroot(MD5: 1f7fed180237ed352d274c69012a4717)的初始加载点。Mebroot是一个主引导记录感染恶意软件，运行在现代操作系统[1]。它的根本目的是从被害者处获取信用卡号和其它财务信息。这也用于在感染的机器上执行下载其它恶意代码。大部分恶意功能在内核模块实施。我们将会使用VERA来分析用户模块家在能力。

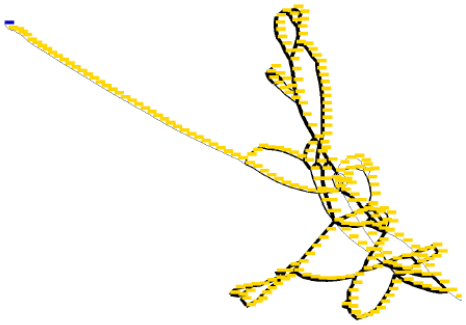

图 5：Mebroot 初始 45 分钟忙碌环

最初运行分析是让Mebroot执行大约5分钟。这个结果在图 5中显示，并且严格限制的。因为显示了很少的信息，程序的这部分最可能是一个忙碌的循环来阻止快速分析，并且通常导致分析到一个错误的路径。通过允许执行12小时，我们能够获得这个执行的木马更加广阔的视图。这显示在图6中。

Mebroot的一个复杂分析工作是它初始加载功能。木马防止自己被分析，通过首先进入到一个忙碌循环并执行将近45分钟。在这期间，不执行任何关键内容。一旦这个延迟完成，Mebroot将会感染主机的主引导记录。主引导记录保存着初始化代码，之后在开机流程执行后将感染运行的Windows内核。

从图表我们能够关联运行地址到反汇编程序IDA中。我们注意到的一个主要特征是一个被称为中间指令指针寄存器的跳转。这个混淆技术依赖于Intel指令集的数据密度。当一个静态逆向分析工具例如IDA分析代码时，指令与那些正在执行的是不一样的。Nick Harbour描述了这个特有的问题在[9]并且与Mebroot内在表现相符合。知道这个Mebroot的特征让我们能用IDA更正它的反汇编代码视图。从分析角度，知道这个把戏非常有用，因为它提供了一个分析中的代码的准确的视图。

图 6：Mebroot 全部执行过程概览

Mebroot的其它部分执行在特权内核空间，并且在这个分析中不明显。我们知道木马基于网络日志成功执行，并且IDS通过签名识别IP地址来判断Mebroot感染。

7. 用户调研

为了评估这个工具和方法对程序分析的效果，我们进行了一个初步的用户调查。事先一个星期前，用户参加了一个逆向训练课程。他们学习了第三章所述的过程，并且都通过了一个这个工程的证书考试。当用户获得一个使用这个工具的介绍和一系列的打印版的 VERA 的说明书后，他们进行了一个典型分析来熟悉这个工具和方法。在这个训练课程中，用户被要求执行一个评估，使用这个 VERA 可视化工具分析两个恶意软件样本。这些样本由两个不同的打包器加密：UPX 和 Mew。

用户明确要求，即在标准评估过程中，他们的工具在每一步骤帮助哪些方面。首先找到一个封装程序的初始进入点。第二执行程序来寻找任何可识别的输出。使用工具来监控程序运行时系统的改变。然后在于程序并且开始逆向工程，解混淆需要用到的代码。最后鉴定主环来显示分析相关和关键代码部分。

在分析过程的每一部分，用户需要反馈他们如何完成工作的每一步，并介绍在这一步中发现了什么。

在分析阶段结束后，用户需要评估使用 VERA 的优点和不足，使用 VERA 是否提高了分析速度，以及他们是否发现了他们认为用传统设备无法发现的东西，或者他们是否丢掉了他们认为用传统设备会发现的东西。最后，用户被问到他们是否想要再次使用 VERA 以及他们是否会向同事推荐 VERA。

成功找到代码的执行部分的用户显示在图 7 中，显示了用户找到初始进入点的数量，初始代码以及展示了主循环。另外，如图所示，所有的用户都说他们还会使用 VERA 并且推荐它。这表示 VERA 从初学者到老手的接受性，即使实际上还需要进行更多的实验来显示统计数据。

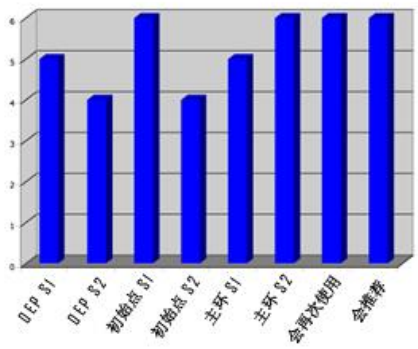

图 7：这个表格展示了使用 VERA 发现两个不同样本特定部分代码的用户数量。OEP 代表初始入口点。这个表格也展示了所有六个用户表示会再次使用 VERA 并且他们会推广 VERA。

所有的用户反馈中，唯一实质上的负面评价是即使用户 1 能够鉴别一个循环的开始，但是他无法鉴别循环的结束；用户表示很多环变得缠绕复杂。

用户 1 说他“能够更快的找到关键部分”。用户而“第一眼就紧紧抓住了主循环，比仅使用 IDA 容易得多”。用户 5 表示“通过执行路径的可视化特征，更容易找到初始进入点”。用户 6 表示“能找到执行了很多次的重要内容”。

用户提出了改进 VERA 的一些有意义的建议。用户 2“希望能够进入内存地址，并看到与地址相关的基本块被突出显示”。用户 3 希望软件能够“隐藏和显示全部或部分环”，与用户 6 的意见类似。

在综合评价中，用户 1 表示“可视化分析很棒，能更好的定位关键部分”，用户 2 说“很好的工具。这个工具有潜力能够大大减少分析时间”，用户 4 说“它很重要。尽快发布。”总的来说，用户调研证明了 VERA 的有效性和易用性。

8. 结论

我们提出的 VERA 框架提供了加速逆向工程的方法。这个工具已经在很多商业和学术工具上使用，来帮助更好地理解一个复杂程序的流和组成。用户研究表明这个工具提高分析并降低了逆向应用程序所需的总体时间。

我们已经改进了 Ether 分析框架更好的使用传统分析技术，包括我们已有的可视化工具。这些已经加入到主线 Ether 系统。

还有一些我们要探索未来工作领域。首先将实施改善对执行程序内部的环的突出显示。这是用户对这个工具的普遍要求。第二，开发一个基于程序分析方法的内核。这是针对现代恶意软件过渡到 Windows 内核的响应。最后将开发一个 3D 显示环境，来提供程序分析的进一步理解。

致谢

作者希望向 Alan Erickson, Cort Dougan, Paul Royal, Artem Dinaburg 和 Moses Schwartz 表示感谢，他们为我们提供了非常有价值的帮助。

参考文献

[1] K. Alkio. Mbr rootkit, a new breed of malware. F-Secure Blog. http://www.f-secure.com/weblog/archives/ 00001393.html.
[2] U. Bayer. Ttanalyze: A tool for analyzing malware. Masters thesis, Technical University of Vienna, 2005.
[3] J. Bohnet and J. Do¨llner. Visual exploration of function call graphs for feature location in complex software systems. In SoftVis ’06: Proceedings of the 2006 ACM symposium on Software visualization, pages 95–104, New York, NY, USA, 2006. ACM.
[4] C. Cifuentes, T. Waddington, and M. V. Emmerik. Computer security analysis through decompilation and high-level debugging. In Eighth Working Conference on Reverse Engineering. IEEE Computer Society Washington, DC, USA, 2001.
[5] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), Nov. 2008.
[6] P. Ferrie. Attacks on virtual machine emulators. Symantec Advanced Threat Research, 2006.
[7] P. Ferrie. Anti-unpacker tricks - part one. Virus Bulletin, 2008.
[8] F. Guo, P. Ferrie, and T.-c. Chiueh. A study of the packer and its solutions. In RAID, Cambridge, Massachusettes.
[9] N. Harbour. Advanced software armoring and polymorphic kung-fu.In Defcon 16, Aug. 2008.
[10] HBGary. Responder professional. Product Description Page. https://www.hbgary.com/products-services/ responder-professional/.
[11] Hexrays. Ida pro disassembler and debugger. Product Description Page. http://www.hex-rays.com/idapro/.
[12] M. G. Kang, P. Poosankam, and H. Yin. Renovo: A hidden code extractor for packed executables. In Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM), Oct. 2007.
[13] J. Ma, J. Dunagan, H. J. Wang, S. Savage, and G. M. Voelker. Finding diversity in remote code injection exploits. In Internet Measurement Conference, Rio de Janeiro, Brazil, 2006. ACM.
[14] N. Nethercote and J. Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), San Diego, CA, 2007. ACM.
[15] D. Quist and V. Smith. Covert debugging: Circumventing software armoring. In Blackhat USA, Aug. 2007.
[16] P. Royal. Alternative medicine: The malware analyst’s blue pill. In Blackhat USA, Aug. 2008.
[17] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. Polyunpack: Automating the hidden-code extraction of unpack-executing malware, 2006.
[18] J. Stewart. Ollybone: Semi-automatic unpacking on ia-32. In Defcon 14, Las Vegas, NV, 2006.
[19] A. Telea and L. Voinea. An interactive reverse engineering environment for large-scale c++ code. In SoftVis ’08: Proceedings of the 4th ACM symposium on Software visualization, pages 67–76, New York, NY, USA, 2008. ACM.
[20] A. Vasudevan and R. Yerraballi. Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation. volume 48, Hobart, Tasmania, Australia, January 2006. Australian Computer Society, Inc.
[21] Y. Xia, K. Fairbanks, and H. Owen. Visual analysis of program flow data with data propagation. In VizSec ’08: Proceedings of the 5th international workshop on Visualization for Computer Security, pages 26–35, Berlin, Heidelberg, 2008. Springer-Verlag.
[22] O. Yuschuk. Ollydbg debugger and disassembler. Product Description Page. http://www.ollydbg.de/.
[23] Zynamics. Binnavi. Company Product Description Page. http://www.zynamics.com/binnavi.html.