决策树方法在恶意 DLL 文件检测中的应用①

Application of Decision Tree in Malicious DLL Detection

夏 丽 袁津生 (北京林业大学 计算机系 北京 100083)

摘 要: 本文对现有恶意 DLL 文件注入技术和 PE 结构进行了分析，提出了一种检测恶意 DLL 文件的新方法。通过分析 DLL 文件的文件属性和 PE 文件字段值，利用决策树方法中的 C4.5 算法构造恶意 DLL 文件检测模型，并且通过实验验证该检测模型的检测效率。

关键词: 恶意 DLL DLL 注入 PE 决策树 C4.5

1 引言

恶意 DLL 文件是由一段实现恶意功能的代码加上一些特殊代码写成的 DLL 文件。恶意 DLL 文件利用这段恶意代码实现木马病毒功能。由于恶意 DLL 文件使用动态链接库技术加载运行，没有独立进程存在，越来越多的恶意攻击使用 DLL 注入技术，把恶意 DLL 文件注入到目标进程的地址空间中。恶意 DLL 文件的危害一方面是它可以同普通恶意文件一样实现各种恶意功能；另一方面是恶意 DLL 文件在运行时不会产生进程，而且通常会被注入到系统进程中，使用传统检测方法很难查杀。

目前主流杀毒软件主要采用复合特征码技术来检测恶意文件。黑客为了防止恶意文件被查杀，通常综合使用加壳、加指令、修改特征码、变换入口地址、入口点加密等免杀方法。本文提出的检测方法不是基于特征码技术，而是在分析 DLL 的文件属性和 PE 文件一些字段值的基础上，从中概括出一些用于检测恶意 DLL 文件的属性，并根据这些属性生成决策树，通过决策树模型判断恶意 DLL 文件。

2 动态链接库(DLL)文件

DLL 文件即动态链接库文件，是一种可执行文件，它允许程序共享执行特殊任务所必需的代码和其他资源。Windows 提供的常见 DLL 文件中包含外部函数和资源。动态链接库是应用程序共享资源、节省内存空间、提高使用效率的一个重要技术手段。

2.1 DLL 注入技术简介

恶意 DLL 文件同正常的 DLL 文件一样不能独立运行，必须由进程加载运行，所以恶意 DLL 文件必须先被目标进程加载后才能执行其恶意功能。DLL 注入技术就实现了把恶意 DLL 文件加载到目标进程的地址空间中。

目前来看，主流的恶意 DLL 文件注入方法主要有两种，一种是 Windows hook(系统钩子)，一种是远程线程。

Windows hook 利用全局钩子(全局钩子的钩子函数必须封装在动态链接库 DLL 中使用)，使其作为其他程序的一部分被加载运行[1]。

远程线程技术是在指定的目标进程中创建线程并且让目标进程中的线程调用 LoadLibrary 函数来加载必要的 DLL[2]。由于远程线程不仅容易实现，而且可以很好的隐藏线程，所以很多的 DLL 木马都采用远程线程技术注入到目标进程的地址空间中。

2.2 PE 文件结构分析

在 Windows 环境下，可执行文件(EXE、DLL 文件等)主要以 PE(Portable Executable)文件格式存在。它是微软制定的可执行的二进制文件格式[3]。

PE 文件中有大量的关于可执行文件的静态信息。研究中我们发现恶意 DLL 文件与合法 DLL 文件在 PE 文件结构中的一些字段值有所不同，我们可以利用这些不同的字段值来检测恶意 DLL 文件。下面简单的介绍一下 PE 文件及其结构。

① 收稿时间:2008-09-10

PE 结构包括 MS-DOS 头、PE 签名、文件头、可选头、节头和各节组成。PE 文件的详细布局如图 1 所示。

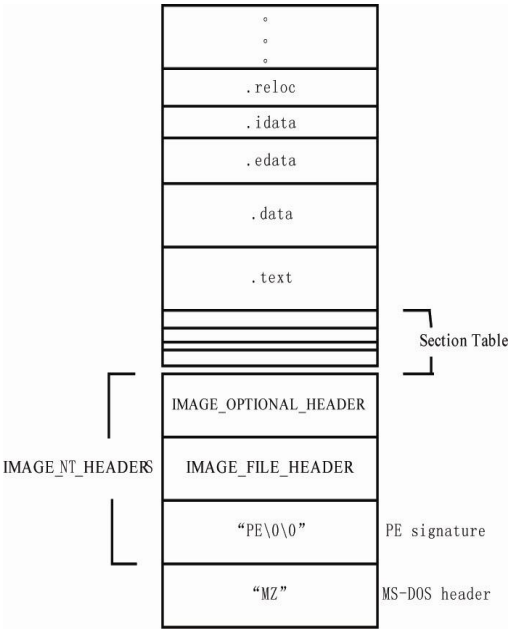

图 1　PE 文件结构

(1) MS-DOS 头。PE 文件都是从 MS-DOS 头开始，其中被称为 MS-DOS stub 小程序功能很简单，在缺省情况下，这个程序只是显示 "The programme cannot be run in MS-DOS mode" .以提示用户该文件在 Win32 下执行。

(2) PE 签名(signature)。它可以将一个 PE 二进制文件和其他含有根的二进制文件区分开来，对于 PE 文件来说，signature 字段值已由#define 定义为 0X00004550 即字母 PE/0/0。

(3) 文件头(IMAGE_FILE_HEADER)。这个部分包含了 PE 文件最基本的信息，用来说明二进制文件运行的机器、区段的个数、链接的时间、可执行文件是 EXE 文件还是 DLL 文件。

(4) 可选头(IMAGE_OPTIONAL_HEADER)。文件头之后附加的信息结构，它包含二进制文件附加的信息，如开始地址、保留堆栈、数据段大小、可选头尾部是一个数据目录，这些目录包含许多指向各节目录的指针。

(5) 节头(Section Table)。描述了每一个节在映像中的信息，节头实际上是一个数组，数组每一项描述了 PE 文件的每一个节的信息。

(6) PE 节(Section)。PE 节可以根据节中包含的内容进行划分。这些节包括数据、代码、资源等信息。PE 文件预定义的节有：.text、.bss、.crt、.rsrc、.idata、.edata、.reloc、.tls、.rdata、.debug，实际上 PE 文件的节数和节名是不确定的，可以根据需要定义节名。

3　基于决策树的恶意DLL文件检测方法设计

本文检测的目的就是希望能够检测出试图注入到进程中的 DLL 文件是否是恶意文件。经过对大量恶意 DLL 文件分析，我们归纳总结出一些用于标识恶意 DLL 文件的属性。本方法设计思想是提取这些属性，然后引入决策树构造检测模型。

3.1 恶意 DLL 文件分析

经过对大量合法 DLL 文件和恶意 DLL 文件的分析实验，本文在 DLL 文件的文件属性和 PE 文件的字段值中选择了相对可以明显检测出恶意 DLL 文件的属性。下面列出了用于检测的属性，这些属性是通过大量的实验总结得出的。

(1) DLL 文件数字签名。数字签名是微软为了保护系统文件而采取的一种技术，一般来说几乎所有的 Windows 系统的 DLL 文件都能通过数字签名，当然不是微软的 DLL 文件也可能有自己数字签名，并且能通过微软的认证。我们在验证 DLL 文件数字签名时遵循三个原则：一是所有拥有合法数字签名的文件都不会对系统产生危害。也就是说通过数字签名的 DLL 文件一定是合法的文件，因为使用数字签名的文件哪怕只改动了该文件的一个字节都不能通过数字签名验证。二是病毒木马等恶意 DLL 文件没有合法的数字签名。三是并不是所有没有数字签名的文件都是有害的。

(2) DLL 的文件属性。DLL 文件一般都有文件属性信息。这些信息包括公司名(company name)、产品名(product name) 、版权、描述、版本号等等。这些信息都存在于 VERSIONINFO 中。下面详细介绍一下我们在检测中将要用到的属性信息。公司名(company name)是为了指明 DLL 文件所属的公司。例如微软的 DLL 文件公司名中一定包含 "Microsoft" 这个字段。产品名(product name)是 DLL 文件所属的产品。例如绝大多数微软系统 DLL 文件在 product name 中包含 "Microsoft operating system" 或 "操作系统" 这样的字段。经实验证实拥有这种字段的合

法 DLL 都能通过数字签名验证。DLL 文件通常都有版权、版本号。微软的 DLL 文件都有版权和版本号。许多恶意 DLL 文件为了实现更好的隐藏，通常修改其文件属性用以伪装成微软系统的 DLL 文件。

(3) Characteristic。PE 文件头中的一个属性，该属性值用来描述一个 PE 文件是一个可执行文件还是一个动态链接库文件。DLL 文件在该字段的 16 进制值高位为 2。

(4) CheckSum。PE 文件的 CRC 校验和，几乎所有微软合法的 DLL 文件都有自己的校验和，一些 DLL 文件不提供校验和并且可能为 0。

(5)可选头中的四个参数。这四个参数位于 PE 可选头中，分别为 Baseofcode、SectionAlignment、FileAlinment、SizeofStackCommit。微软 DLL 文件的这四个参数值是固定的，对应 16 进制值分别为 0X1000、0X1000、0X200、0X1000。

(6) Recourse。DLL 文件的资源，比如对话框、菜单、图标等等，DLL 文件通常都包含资源。

(7) Section name。PE 文件的节名可以根据需要自定义。微软 DLL 文件有自己预定义的节名。经过对微软 DLL 文件的大量实验，微软 DLL 文件所用的节名只有预定义的节名另外还可能有 orpc、ExtraDat 和 Shared 节。不是微软的 DLL 文件也通常使用微软的预定义节名，但他们也可能有自定义的节名。一些通过生成恶意 DLL 文件的软件产生的恶意文件和利用软件进行免杀处理的恶意 DLL 文件通常会在 PE 节上添加标志性的名称。例如经过 UPX 加壳处理的程序会在 PE 节上添加 upx0，upx1 等标志性的名称，经过 MaskPE 加密处理(打乱程序的源代码进行免杀)的程序会在 PE 节上添加 MaskPE 名称。如果存在这样的名称，我们可以基本上把它归结为恶意 DLL 文件，因为正常的系统文件及一些第三方的软件都很少加壳压缩，而恶意文件如病毒木马几乎都采用这样的处理。

3.2 决策树方法

决策树方法是一种通过构造决策树来发现训练集中分类知识的数据挖掘的方法。决策树是一棵树，内部节点是分裂属性，叶节点是类别值。构造决策树分为两步：① 决策树的生长：由训练样本集生成一棵决策树；②剪枝：用非训练集中的实例检查生成的决策树，剪去影响精度的分枝[4]。

经过 2.1 节中对标识恶意 DLL 文件的属性分析，为了构造一个合理的检测模型，本文引入决策树方法。因为我们检测的最终目的是判断 DLL 文件是否是恶意文件，所以我们使用的检测模型要针对不同的检测结果有相应的结果类别的划分。在求解分类问题的方法中，决策树是最有用的一种方法[5]。决策树不仅能创建结构优化，易于理解的分类模型，而且其较高的分类精度可以做到高效的分类预测。决策树所具有的这些优点非常适合应用到入侵检测中。因此本文选择决策树来构造检测模型。

C4.5 算法是发展的比较完善，而且也是比较简单易懂的一种决策树算法。C4.5 算法从树的根节点处的所有训练样本开始，依据信息增益率选取一个属性来区分这些样本。选择信息增益率最大的属性，作为当前的属性节点。属性的每一个值产生一个分支，分支属性值的相应样本子集被移到新生成的子节点上，这个算法递归地应用于每个子节点上，直到节点的所有样本都划分到某个类中[4]。

C4.5 算法计算信息增益率公式为

$$GainRatio(D,S)=\frac{Gain(D,S)}{H\left(\frac{|D_1|}{|D|}\cdots\cdots\frac{|D_s|}{|D|}\right)}$$

其中，$Gain(D,S)$ 是分裂的信息增益，$H\left(\frac{|D_1|}{|D|}\cdots\cdots\frac{|D_s|}{|D|}\right)$，是分裂属性的熵[5]。

生成决策树后，本文采用事后修剪的方法进行剪枝，该方法当决策树充分生长完成后，修剪掉多余的树枝[6]。

3.3 基于决策树构造检测模型

我们采用 C4.5 算法，在生长树的每一步选择信息增益率最高的属性作为当前节点的测试属性，最后得出决策树模型。决策树的生成过程如下：

(1)选择分裂属性和建立分类结果

依据 2.1 对恶意 DLL 文件属性的分析，本文选择了以下属性作为分裂属性。

① DLL 文件是否通过数字签名验证。

② product name 不包含“operating system”或“操作系统”这样的字段。

③ characteristic 是 0x2*，即 16 进制值的高位值是否为 2。

④ company name 是否有“Microsoft”这个字段。

⑤ 是否满足微软 DLL 固定参数（包括版本号、

版权是否存在，是否满足可选头中的四个参数值)。

⑥ checksum 值是否存在。

⑦ resource 是否存在。

⑧ Section name 是否存在恶意节名，本文通过对大量恶意 DLL 文件 PE 节名的分析，总结出一些经常出现的并且能够证明是恶意 DLL 文件的 PE 节名，简称为恶意节名。这些经常出现的恶意节名有 Upack、Aspack、UPX、sdt、MaskPE。这些节名都是由于使用生成恶意 DLL 文件的软件和免杀软件才添加到 PE 节上的。

前 7 个分裂属性的分裂值是二元属性值分别为“是”和“否”，最后一个分裂属性的分裂值是三元属性值，它的取值是字符串类型，分别是微软常用节名、自定义的节名(不含恶意节名)、恶意节名。

我们把 DLL 文件的分析结果划分成两类，分别是类别 1 合法 DLL 文件，类别 2 恶意 DLL 文件。

(2) 准备训练数据

决策树模型的生成是以训练数据为基础的，我们构建决策树所要用到的训练数据来源有两种。一是长期数据的积累，一是专家构建的训练数据。其中数据的积累是训练数据的主要来源。我们把这些数据组合成训练数据集，训练数据集是由一组记录构成，每条记录由用于检测的属性值和记录末尾类别标记构成。表 1 为训练数据的一个简要示例。

表 1　决策树训练数据样例表

属性 / 类别	数字签名	Product name	Characteristics (16进制值	DLL 固定参数 (16进制值	Checksum (16进制值	Resource(大小 (16进 制值	Section name
合法 DLL	通过	Microsoft	210E	微软 4个参数值	000A6BFE	00028878	微软节名
恶意 DLL	未通过	无	210E	微软 4个参数值	无	无	有UPX节

(3) 由信息增益率选择分裂属性的次序创建决策树

为了选择分裂属性的最佳次序，我们先计算所有分裂属性的信息增益率，选择信息增益率最大的作为根结点，剩余属性依照上面步骤进行进一步划分，如果划分到最终的类别或没有属性可以划分则为叶结点。

我们通过上述过程生成了决策树，并且把生成的决策树经过修剪得到的最终决策树如图 2 所示。

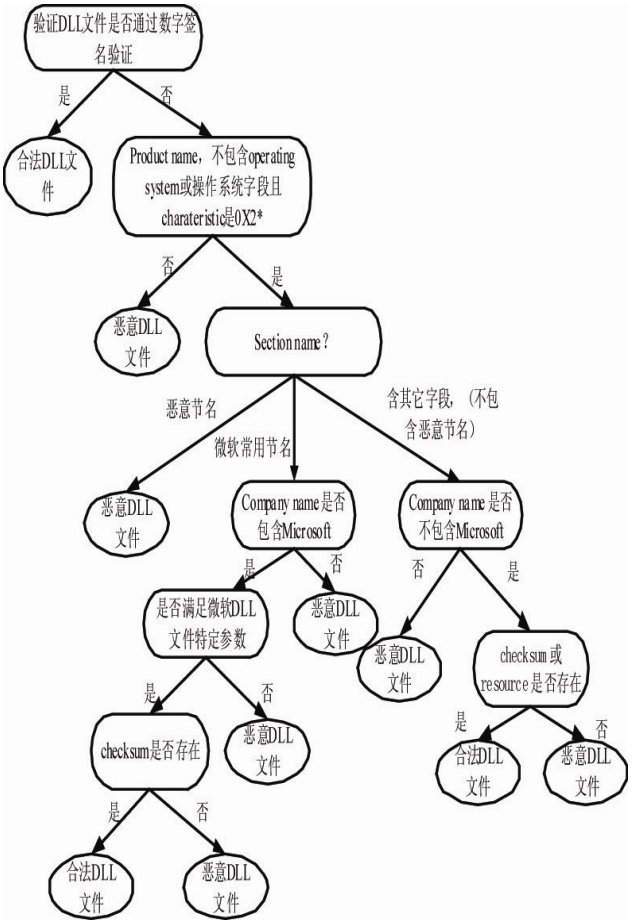

图 2　检测恶意 DLL 文件决策树结构模型

4　决策树检测模型验证及实验结果

为了验证恶意 DLL 文件决策树检测模型的效率，本文构建了一个简单的检测系统。

4.1 检测系统结构

当给定一个需要检测的 DLL 文件时，本文采用的检测系统结构如图 3 所示。

本文对检测的 DLL 文件首先依照训练数据中的属性提取 DLL 文件中的各属性值，然后通过检测引擎(遍历决策树模型)进行检测，最后响应检测结果。

图 3　检测系统结构图

4.2 实验结果

本文随机选取了 212 个恶意 DLL 文件，又从合法文件中随机选取了 1385 个合法文件，把这 1597 个文件混合在一起作为检测数据集，然后用本文的检测系统进行检验。

本文使用漏报率和误报率两个指标来衡量模型的检测效率。检测结果如表 2 所示。

表 2　恶意 DLL 文件检测结果

检测结果 / 类别	合法文件	恶意文件
恶意 DLL 文件（212个）	11	201
合法 DLL 文件（1385个）	1352	33

由表 2 可以看到恶意 DLL 文件检测漏报率为 5.2%，合法 DLL 文件的检测误报率为 2.4%。从实验结果看出，本文检测的漏报率和误报率不仅在可接受的范围之内，而且同目前能够检测恶意 DLL 文件的检测工具相比较，本文的检测模型有很高的检测效率。

4 结束语

本文提出了一种通过分析 DLL 文件来检测 DLL 文件是否是恶意文件的方法，根据 DLL 文件的相关属性采用决策树构造检测模型，实验表明这种方法有很高的效率。漏报率<6%，误判率<3%。

目前这种检测方法对绝大多数恶意 DLL 文件的检测是可行的，但是对于误报和漏报的 DLL 文件，我们仍需要进一步研究 DLL 文件和相应 PE 文件的属性来提高我们的检测效率。

参考文献

1 余美德,于志平.Windows 钩子技术在病毒程序中的应用.现代计算机, 2005,(2):83－86.

2 Richter J.　Windows 核心编程,北京:机械工业出版社, 2000:531－534.

3 伍红兵.Visual C++编程深入引导.北京:中国水利水电出版社, 2002:440－474.

4 刘苇,张永平.决策树算法在入侵检测中的应用分析与改进.计算机工程与设计, 2006,(19):3641－3643.

5 Dunham MH.数据挖掘教程.北京:清华大学出版社, 2005:83－86.

6 魏晓云.决策树分类方法研究.计算机系统应用, 2007,16(9):42－44.