基于 K-最近邻算法的未知病毒检测

张波云[1,2]　殷建平[1]　张鼎兴[1]　嵩敬波[1]
[1](国防科技大学计算机学院,长沙 410073)
[2](湖南公安高等专科学校计算机系,长沙 410138)

摘　要　因为准确检测计算机病毒是不可判定的,故该文提出了一种基于实例学习的 k-最近邻算法来实现对计算机病毒的近似检测。该法可以克服病毒特征代码扫描法不能识别未知病毒的缺点。在该检测方法的基础上,文章设计了一个病毒检测网络模型,此模型适用于实时在线系统中的病毒检测,既可以实现对已知病毒的查杀,又可以对可疑程序行为进行分析评判,最终实现对未知病毒的识别。

关键词　计算机病毒　K-最近邻算法　病毒检测

文章编号 1002-8331-(2005)06-0007-04　**文献标识码** A　**中图分类号** TP309.05

Unknown Computer Virus Detection Based on K-Nearest Neighbor Algorithm

Zhang Boyun[1,2]　Yin Jianping[1]　Zhang Dingxing[1]　Hao Jingbo[1]
[1](School of Computer Science,National University of Defense Technology,Changsha 410073)
[2](Dept.Computer Science,Hunan Public Security College,Changsha 410138)

Abstract: Because precise determination of a virus by its appearance is undecidable,a K-nearest neighbor Algorithm based on sample learning to detect computer virus approximately is presented in this paper. It can overcome the shortage of normal virus scanner -which can not detect unknown virus. Based on this method,a virus detect network model is designed also.This model fits to detect viurs in the on-line system,it alse detect known and unknown computer virus by analyzing the program's behavior.

Keywords: computer virus,K-Nearest Neighbor Algorithm(KNN),virus detection

1　引言

Fred Cohen 博士在上世纪 80 年代指出了"计算机病毒检测的不可判定性"[1],即"精确检测计算机病毒是不可判定的"。此后 Diomidis Spinellis 证明了"有界长度病毒的可靠检测是一个 NP 完全问题"[2]。因此长期以来,各反病毒研究机构都在努力探讨病毒检测的近似算法,在 IBM 病毒研究中心,曾经成功地将神经网络用于判断引导型病毒[3],他们声称在不久的将来把人工免疫方法应用于病毒检测产品中去。基于上述思想的启发,该文采用一种基于实例学习的 k-最近邻算法 (K-Nearest Neighbor Algorithm,简记为 KNN)[4]来实现对计算机病毒的检测。

当前的计算机病毒检测技术主要基于特征检测法,其基本方法是提取已知病毒样本的特征,并将此特征数据添加到病毒特征库中,在病毒检测时通过搜索病毒特征库查找是否存在相匹配的病毒特征来发现病毒。这种检测办法只能用于检测已知的病毒,对于新出现的病毒的检测无能为力。为了解决这一问题,文章提出一种采用基于实例学习的 k-近邻算法对提取的可疑文件行为特征进行分析,并利用病毒程序与正常程序的行为特征的差异性进行分类,从而达到检测未知病毒的目的。在该检测方法的基础上,该文设计了一个恶意代码检测网络模型,从模型的设计和技术实现的角度进行分析,介绍了它的工作原理。这种模型十分适用于实时不停机系统中的未知病毒检测,既可以实现对已知病毒的查杀,又可以对可疑文件进行分析评判,最终实现对未知病毒的识别。最后对模型的功能进行了实验测试,结果达到了很好的实际应用效果。

2　模型结构分析

系统结构框图如图 1 所示。系统由病毒防火墙、服务器和病毒检测服务器组成,进入系统的可疑文件首先经过系统的第一道防线病毒防火墙的检测,若没有检测出病毒,则将其复制两份,一份拷贝直接存放于 Server 中;一份拷贝则进入基于 KNN 的 Detect Server 中,对其行为特征进行检测,如果发现其可疑行为,判断为感染了未知病毒,Detect Server 则将信息反馈给 Server,在 Server 中对该文件进行隔离、监管或删除。由于基于程序行为的杀毒系统可能引发对系统不可预料的破坏,故在该系统中配置了专门进行未知病毒检测的 Detect Server,它对正常服务器的性能影响极微,十分适用于实时在线系统中的未知病毒检测。而对于 Detect Server 自身的安全防护,可以采用自还原保护法等。

系统的逻辑结构如图 2 所示。Detect Server 中的病毒检测引擎采用 K-最近邻算法实现。第一阶段,系统首先进行自我训练,采用特征提取工具对样本空间 U 中的训练样本程序进行

基金项目:国家自然科学基金项目(编号:60373023)资助
作者简介:张波云(1972-),男,博士生,研究方向为计算机病毒、网络安全等。殷建平,教授,博士生导师,研究方向为算法设计与分析、人工智能、模式识别、信息安全等。

特征提取，并保存在训练样本特征库中以备将来学习和检测病毒之用。第二阶段，当外来文件进入系统时，先经过病毒防火墙的第一次测试，若发现其已染有病毒，则在 Server 中对其隔离，保存入病毒隔离库中；若没发现病毒，则在 Detect Server 中继续调用 KNN 检测引擎测试。将可疑程序特征提取出来以获取被检测文件的特征矢量，然后用 KNN 算法对可疑程序进行病毒判定。若判断为病毒，则将信息反馈给 Server，在 Server 中对病毒文件进行隔离、监管或删除，也可以将其发送给相关专家进行进一步精确分析。与此同时把新病毒样本添加到训练样本空间，重复上述过程进行新一轮的学习。

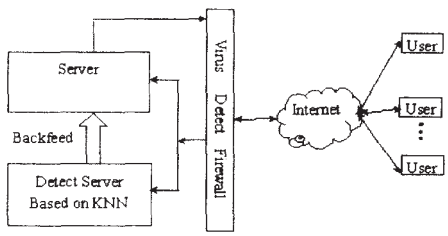

图 1　系统结构框图

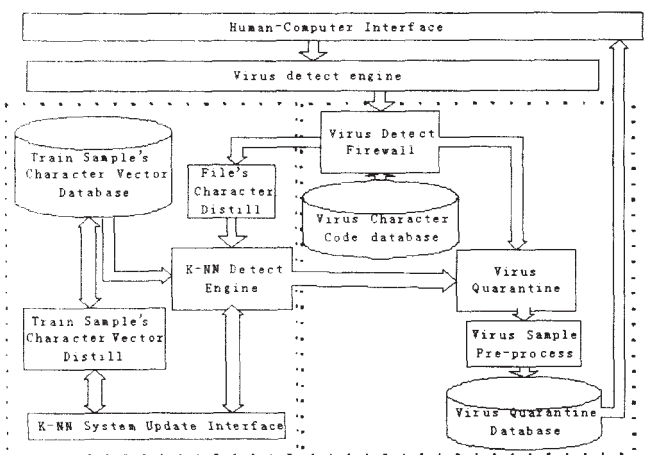

图 2　基于 k-近邻算法的病毒检测系统逻辑流程

3　k-最近邻算法数学模型

用最近邻方法进行预测的理由是基于假设“近邻的对象具有类似的预测值”。最近邻算法的基本思想是在多维空间 $\Re^n$ 中找到与未知样本最近邻的 k 个点，并根据这 k 个点的类别来判断未知样本的类。这 k 个点就是未知样本的 k-最近邻。算法假设所有的实例对应于 n 维空间中的点。一个实例的最近邻是根据标准欧氏距离定义，设 x 的特征向量为：

$$<a_1(x),a_2(x),\cdots,a_n(x)>$$

其中 $a_r(x)$ 表示实例 x 的第 r 个属性值。两个实例 x_i 和 x_j 间的距离定义为 $d(x_i,x_j)$，其中：

$$d(x_i,x_j)=\sqrt{\sum_{r=1}^{n}(a_r(x_i)-a_r(x_j))^2}$$

在最近邻学习中，离散目标分类函数为 $f:\Re^n\to V$，其中 V 是有限集合 $\{v_1,v_2,\cdots,v_s\}$，即各不同分类集。最近邻数 k 值的选取根据每类样本中的数目和分散程度进行的，对不同的应用可以选取不同的 k 值，一般 $k=\lfloor\sqrt{n}\rfloor$，式中 n 为特征矢量个数。

如果未知样本 s_i 的周围的样本点的个数较少，那么该 K 个点所覆盖的区域将会很大，反之则小。因此最近邻算法易受噪声数据的影响，尤其是样本空间中的孤立点的影响。其根源在于基本的 k-最近邻算法中，待预测样本的 k 个最近邻样本的地位是平等的。在自然社会中，通常一个对象受其近邻的影响是不同的，通常是距离越近的对象对其影响越大。故在系统中采用距离加权最近邻算法，在 k-最近邻算法中引入对象间的距离概念，对 k 个近邻的贡献进行加权，距离较近的对象权值较大，即：

$$f(s_i)=\max\sum_{p=1}^{r}\sum_{j=1}^{k}w_j\delta(v_p,f(s_j))$$

式中 w_j 表示近邻 s_j 的距离权重。权重 w_j 有许多不同的试验方法，最普遍使用的是距离平方的倒数，即 $w_j=\frac{1}{|s_i-s_j|^2}$。

4　未知病毒检测

4.1　病毒特征提取

病毒与一般程序的区别在于执行了一些特殊的动作来破坏系统。从外在表现来看，如木马程序常进行如下的操作：对 win.ini 文件夹特定项的修改；对 system.ini 文件的特定项的修改；对注册表特定键值的修改，文件关联；端口异常与此端口上的数据流量；硬盘数据共享；远程文件操作；远程抓屏操作；计算机关闭与启动；鼠标键盘的操纵；远程执行可执行程序；消息发送；进程管理；更改服务；文件加壳；修改注册表；文件打开；文件复制；文件修改等。

不管是二进制可执行病毒、脚本病毒还是宏病毒，它们都是一种程序，它需要调用操作系统提供的各种功能函数才能达到传播自身和破坏系统的目的。因此可以将下列信息作为待检测程序的行为特征：(a)程序使用的动态链接库(DLL)文件；(b)程序在所使用的 DLL 中调用的 API 函数[5]；(c)程序的机器代码序列或源代码等。

在试验过程中，选取的病毒属性如表 1 所示。并对不同的程序的行为特征划分为 3 个危险级别：即一般、较严重、严重，在图中分别用☆、☆☆、☆☆☆表示。

然后，编写了相应的特征提取工具对待检测程序的上述特征进行获取。

4.2　病毒检测引擎

通过搜集大量的同类程序可执行代码可以构成样本空间 U，利用现有的病毒检测工具对每一个程序进行准确的归类为两种类型：病毒程序和普通正常程序。将样本空间划分为训练集 $S_{training}$ 和测试集 S_{test} 两个集合(训练集和测试集为样本空间中两个不相交的子集 $S_{training}\cap S_{test}=\varnothing$)。另将训练集划分为病毒程序集 S_{virus} 和普通程序集 S_{normal}，且 $S_{virus}\cap S_{normal}=\varnothing$，$S_{virus}\cup S_{normal}=S_{training}$，然后根据病毒类型构造相应的特征提取工具对程序进行特征提取，并应用 KNN 算法对训练集进行分类。同时，当新的病毒样本添加到样本空间后。可以重复上述过程进行新一轮的学习。

计算机病毒检测是一个二值分类问题，即病毒与非病毒两类。对二值分类，若已知 $N(N$ 充分大$)$个训练样本，对于一个新的样本 S_i，其 K 个最近邻样本中 k_1 个样本的值为 1，k_2 个样本的值为 -1，则根据基本的 K-最近邻算法：

$$f(s_i)=\begin{cases}1 & k_1>k_2\\ -1 & k_1<k_2\end{cases}\qquad k=k_1+k_2$$

由上式可以断定样本 S_i 是否是病毒。一个简单的示例如图 3 所示，图中“+”表示正常程序，“－”表示病毒程序。现正常

表 1　病毒属性列表

序号	程序行为	相关 API 调用	危险度	动态链接库
1	堆操作	RtlFreeHeap ;RtlAllocateHeap	☆☆	NTDLL.DLL
2	动态库加载与释放	LoadLibraryA ;FreeLibrary ;GetModuleHandleA ;GetModuleFileNameA	☆	
3	API 地址获取	GetProcAddress	☆	
4	进程操作	OpenProcess ;CloseHandle ;CreateRemoteThread ;ExitThread ;WaitForSingleObject ;CreateThread	☆☆	
5	内存读写	VirtualAlloc ;GetProcessHeap ;VirtualAllocEx ;WriteProcessMemory ;VirtualFreeEx ;OpenMutexA ;		
		CreateMutexA ;VirtualProtect ;VirtualFree ;HeapFree	☆☆	
6	读注册表	RegCloseKey ;RegEnumKeyExA ;RegOpenKeyExA ;RegQueryValueExA ;RegNotifyChangeKeyValue ;	☆	
7	写注册表	RegSetValueExA ;RegCreateKeyExA ;RegDeleteKeyA	☆☆	
8	程序执行	WinExec ;CreateProcess	☆☆	KERNEL32.dll
9	文件读 & 创建	CreateFileA ;ReadFile ;OpenFile	☆	
10	文件写	WriteFile ;WriteFileEx	☆☆	
11	文件删除	DeleteFileA	☆☆☆	
12	文件移动	MoveFileA ;MoveFileExA ;	☆☆	
13	变更属性权限	SetFileAttributesA ;SetFileTime	☆☆	
14	文件搜索	FindClose ;FindFirstFileA ;FindNextFileA ;FindResourceA	☆☆☆	
15	目录搜索	GetWindowsDirectoryA ;GetSystemDirectoryA ;SetCurrentDirectoryA ;CreateDirectoryA	☆☆☆	
16	目录删除	RemoveDirectoryA	☆☆☆	
17	磁盘操作	GetDiskFreeSpaceA ;GetDriveTypeA	☆	
18	时间操作	GetTickCount ;GetSystemTime ;GetTickCount ;GetLocalTime	☆	
19	系统重启	Exitwindows ;ExitwindowsEx ;AbortSystemShutdown ;InitialteSystemShutdown	☆☆	
20	加密与解密	CryptAcquireContextA ;CryptGenKey ;CryptDestroyKey ;CryptImportKey ;CryptExportKey ;CryptEncrypt ;		
		CryptDecrypt ;CryptReleaseContext ;	☆☆☆	ADVAPI32 .dll
21	远程通讯	WSAStartu ;Socket ;Connect ;Recv ;Send ;Bind ;Listen ;Accept ;Gethostname ;Gethostbyname ;		
		Closesocket ;WSACleanup	☆☆	WSOCK32.dll
22	进程搜索	EnumProcesses ;EnumProcessModules ;GetModuleBaseNameA	☆☆	PSAPI.dll
23	注册表操作二	SHDeleteValueA ;SHGetValueA ;SHSetValueA	☆☆	SHLWAPI.DLL

程序数为 3,病毒程序数为 2,3>2 故可疑程序 x_q 判断为正常。

考虑到算法的稳定性,实验中使用的是距离加权法,详细算法如表 2 所示。

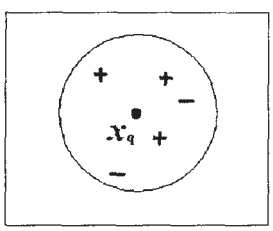

图 3　近邻数 K=5 时的一个示例

表 2　基于距离加权 KNN 的病毒检测算法

(1)训练算法:

对于每个训练样例<x,$f(x)$>,把这个样例加入列表 *training-examples*

(2)分类算法:

①给定一个要分类的查询实例 x_q

②在 *training-examples* 中选取最靠近 x_q 的 k 个实例,并用 $x_1,x_2,\cdots,x_k$ 表示

③返回

$$\hat{f}(x_q)\leftarrow\max\sum_{p=0}^{l}\sum_{i=1}^{k}\omega_i\delta(v_p,f(x_i))$$

其中,如果 $a=b$,那么 $\delta(a,b)=1$,否则 $\delta(a,b)=0$,式中 $w_j=\dfrac{1}{|s_i-s_j|^2}$

5　实验分析

用于实验的样本数据如表 3 所示。样本空间中样本总数为 632,分为正常程序与染毒程序。正常程序从操作系统平台中选取,笔者选用的是 windows 2000 server 首次安装后,机器中的全部 PE 文件共 423 个。通过各种途径收集的病毒程序 209 个,病毒名单见附录。为获得样本空间中程序的特征向量,笔者编写了 API 调用跟踪器,可以实现 Windows 2000Server 环境下的全部 API 函数调用的拦截。

表 3　用于实验的样本数据

	样本空间	训练集	测试集
正常程序	423	373	50
染毒程序	209	159	50
合　计	632	532	100

实验目标主要是对 KNN 查病毒引擎的错误率进行测试。文章将错误类型分为两种,即(1)将正常程序判断为病毒,称为 False Negative,(2) 将病毒程序判断为正常程序, 称为 False Positive。实验结果见图 4,它反映了 KNN 算法中不同最近邻数 K 对错误率的影响。系统在 K=25 时的 False Negative error rate 最小,约为 4.8%。K 的取值也并不是越大越好,它与特征向量的个数有关。

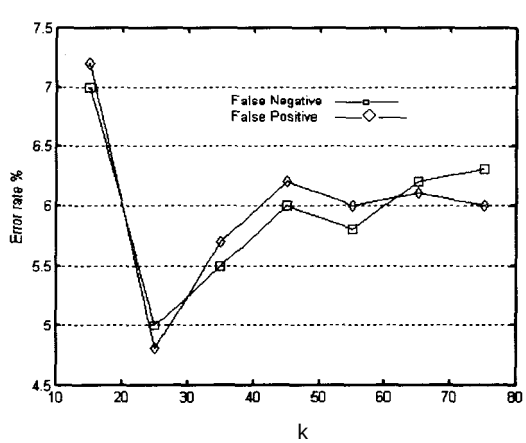

图 4　最近邻数 K 与错误率的关系

6 结论

由于计算机病毒检测的不可判定性，该文尝试用 KNN 算法来实现对计算机病毒的检测，在此基础上，设计了一个恶意代码检测网络模型，较适用于实时在线系统中的未知病毒检测。

此检测模型结合了特征码检测技术与机器学习技术，检测成功率较高。因为 KNN 算法的高效性，它的学习时间几乎为零，系统开销主要集中于程序特征的提取和特征向量间距离的计算，总体上该系统比采用其它机器学习算法的病毒扫描器的效率要高。

该系统是一个病毒检测器，只能隔离病毒而不能实现对病毒的清除，与其它反病毒工具的集成是一个可行的解决方案。因为基于病毒行为的检测会对系统造成不可预料的破坏性，故如何寻找高效与安全的特征提取工具是以后研究的要点。

（收稿日期：2005 年 1 月）

参考文献

1.F Cohen.Computer Viruses：Theory and Experiments[J].Computers & Security，1987，6(1)：22~35

2.Diomidis Spinellis.Reliable Identification of Bounded-LengthViruses Is NP-Complete [J].IEEE TRANSACTIONS ON INFORMATION THEORY，2003，49(1)：280~284

3.Gerald J Tesauro，Jeffrey O Kephart，Gregory B Sorkin.Neural networks for computer virus recognition[J].IEEE EXPERT，1996，(8)：5~6

4.Tom M Mitchell.Machine Learning[M].New York：McGraw-Hill，1997

5.David J Kruglinski，Scot Wingo，George Shepherd..Programming Visual C++[M].Washington：Micosoft Press，1998

附录 样本空间中的病毒集

（病毒样本共 209 个，所有病毒均通过 KasperskyAnti-Virus On-Demand Scanner 检测）

Win32.Adson.1559	Win32.Flechal	Win32.Kala.7620	Win32.Orez.6291	Win32.Slow.8192.a
Win32.Aidlot	Win32.Fosforo.a	Win32.Kanban.a	Win32.Oroch.5420	Win32.Small.1144
Win32.Akez	Win32.Freebid	Win32.Kaze.2056	Win32.Padic	Win32.Spelac.1008
Win32.Alcaul.a	Win32.FunLove.4070	Win32.Kenston.1895.a	Win32.Paradise.2168	Win32.Spit.a
Win32.Aldebaran.8365.a	Win32.Gaybar	Win32.Ketan	Win32.Parite.b	Win32.Staro.1538
Win32.Aliser.7825	Win32.gen	Win32.Kiltex	Win32.Parvo	Win32.Stepar.b
Win32.Alma.2414	Win32.Genu.c	Win32.Klinge	Win32.Peana	Win32.Stupid.a
Win32.Andras.7300	Win32.Ghost.1667	Win32.KME	Win32.Perrun.a	Win32.Taek.1275
Win32.AOC.3649.a	Win32.Ginra.3334	Win32.KMKY	Win32.PGPME	Win32.Tapan.3882
Win32.Apathy.5378	Win32.Ginseng	Win32.Knight.2350	Win32.Pilsen.4096	Win32.This31.16896
Win32.Apparition.a	Win32.Giri.4937.b	Win32.Koru	Win32.Positon.4668	Win32.Thorin.b
Win32.Arianne.1052	Win32.Gloria.2928	Win32.Kriz.3660	Win32.Projet.2404.b	Win32.Tinit.a
Win32.Aris	Win32.Glyn	Win32.Kuto.2058	Win32.Qozah.3361	Win32.Tirtas.5675
Win32.Artelad.2173	Win32.Gobi.a	Win32.Ladmar.2004	Win32.RainSong.4266	Win32.Tolone
Win32.Asorl.a	Win32.Godog	Win32.Lamchi.a	Win32.Ramdile	Win32.Tosep.1419
Win32.Atav.1939	Win32.Halen.2593	Win32.Lamebyte	Win32.Razenya	Win32.Trion.a
Win32.Atav.2073	Win32.Haless.1127	Win32.Lamewin.1751	Win32.Redart.2796	Win32.Ultratt.8152
Win32.Auryn.1155	Win32.Hatred.a	Win32.Lamicho.a	Win32.Redemption.a	Win32.Undertaker.4883.a
Win32.utoWorm.3072	Win32.Henky.504	Win32.Lanky.3153	Win32.Refer.2939	Win32.Vampiro.a
Win32.Awfull.3571	Win32.Heretic.1986	Win32.Lash.a	Win32.Resur.a	Win32.VCell.3504
Win32.Bakaver.a	Win32.Hidrag	Win32.LazyMin.31	Win32.Revaz	Win32.VChain
Win32.Barum.1536	Win32.Highway.b	Win32.Legacy	Win32.Rever	Win32.Velost.1186
Win32.Bee	Win32.HIV.6386	Win32.Levi.3236	Win32.Rhapsody.2619	Win32.Viset.b
Win32.Beef.2110	Win32.HLL.Fugo	Win32.Libertine.d	Win32.Riccy.a	Win32.Vorcan
Win32.Belial.2609	Win32.HLLC.Winatch	Win32.Magic.1590	Win32.Rivanon	Win32.Vulcano
Win32.Belial.a	Win32.HLLO.52736	Win32.Matrix.817.a	Win32.Ruff.4859	Win32.Wabrex.a
Win32.Belial.b	Win32.HLLP.Thembe	Win32.Mauz.a	Win32.Rufoll.1432	Win32.Wanhope.1834
Win32.Belod.a	Win32.HLLP.VB.a	Win32.Mental	Win32.Rutern.5244	Win32.Weird.c
Win32.Bender.1363	Win32.HLLP.Vedex.b	Win32.Miam.5164	Win32.Sadon.900	Win32.Wide.b
Win32.Benny.3219.a	Win32.HLLW.Xelif	Win32.Milen.3205	Win32.Sandman.4096	Win32.Wit.a
Win32.Bika.1906	Win32.Hortiga.4800	Win32.Minit.b	Win32.Sankei.1062	Win32.Wolf.b
Win32.BingHe	Win32.Htrip.a	Win32.MircNew	Win32.Santana.1104	Win32.Xorala
Win32.Blakan.2016	Win32.Htrip.c	Win32.Mix.1852	Win32.Savior.1680	Win32.Yasw.924
Win32.Emotion.a	Win32.Idele.2108	Win32.Mockoder.1120	Win32.Score.3072.a	Win32.Yerg.9412
Win32.Enar	Win32.Idyll.1468	Win32.Mogul.6800	Win32.Segax.1136	Win32.Younga.2384.a
Win32.Enerlam.a	Win32.IKX	Win32.Mooder.g	Win32.Sentinel.a	Win32.Zawex.3196
Win32.Eva.a	Win32.Insom.1972	Win32.Mystery.2560	Win32.Seppuku.2763	Win32.ZHymn.a
Win32.Evar.3587	Win32.Intar.1151	Win32.Nicolam	Win32.Shaitan.3550	Win32.ZHymn.Host
Win32.Evol.a	Win32.InvictusDLL.101.a	Win32.Noise.410	Win32.Shan.1842	Win32.ZMist
Win32.Evul.8192.b	Win32.InvictusDLL.b	Win32.Nox.2290	Win32.Shown.538	Win32.Zombie
Win32.Evyl.b	Win32.Ipamor.a	Win32.Opdoc.1204	Win32.Silcer	Win32.ZPerm.a
Win32.Fighter.b	Win32.Jethro.5657	Win32.Oporto.3076	Win32.Slaman.i	