广义病毒的形式化定义及识别算法

何鸿君[1] 罗 莉[1] 董黎明[2] 何修雄[1] 侯方勇[1] 钟广军[1]

[1](国防科学技术大学计算机学院 长沙 410073)

[2](后勤指挥学院 北京 100858)

摘 要 恶意软件的定义是多年来安全领域的研究重点.恶意软件包括病毒、蠕虫和木马.目前仅有病毒的形式化定义,蠕虫、木马没有公认的形式化定义.按照传统病毒的定义,不存在准确识别病毒的算法.文中提出代码是否为病毒是相对于用户而言的,给用户带来损害的代码才是病毒.据此观点,文中以用户意愿为标准,将病毒区分为显式病毒、隐式病毒,并给出了显式病毒的形式化定义和识别算法.理论分析表明,传统病毒以及大部分木马、蠕虫均属于显式病毒,实际案例分析也证实了这一点.

关键词 病毒;蠕虫;木马;用户意愿;显式病毒;隐式病毒

中图法分类号 TP309 **DOI 号**: 10.3724/SP.J.1016.2010.00562

Formal Definition of Generalized Virus and Its Identifying Algorithm

HE Hong-Jun[1] LUO Li[1] DONG Li-Ming[2] HE Xiu-Xiong[1] HOU Fang-Yong[1] ZHONG Guang-Jun[1]

[1](*Computer Department, School of Computer, National University of Defense Technology, Changsha* 410073)

[2](*Logistics Command Colleges, Beijing* 100858)

Abstract The definition of malicious software is a hot in security domain. Malicious software includes virus, worm and Trojan horse. There is now only formal definition of virus, and no widely accepted formal definitions of worm and Trojan horse. According to definition of traditional virus, there is no algorithm to identify virus definitely. This paper proposes that whether a program code is virus is relative to user, and only those bringing damage to user are viruses. The paper distinguishes viruses to explicit virus and hidden virus based on user's intention, and presents a formal definition of explicit virus and its identifying algorithm. Both theoretical analysis and actual cases study indicates that traditional virus, most of worm, and Trojan horse are explicit viruses.

Keywords virus; worm; Trojan horse; user's intention; explicit virus; hidden virus

1 引 言

病毒的定义是反病毒技术研究的基础,一直是计算机安全领域的重要研究课题,相关论文很多[1].广义病毒包括了传统病毒、蠕虫、木马等所有恶意程序,本文的讨论针对广义病毒.

公认的传统病毒定义是 Cohen 博士 1984 年提出的,"计算机病毒是一种程序,它可以感染其它程序,感染的方式为在被感染程序中加入计算机病毒的一个副本,这个副本可能是在原病毒基础上演变过来的"[2].其后,很少提出新的有影响力的形式化定义[3].

收稿日期:2007-01-03;最终修改稿收到日期:2009-05-28.本课题得到国家"八六三"高技术研究发展计划项目基金(2009AA01Z428)资助.**何鸿君**,男,1968 年生,博士,副教授,主要研究兴趣为信息安全、软件工程经济学等. E-mail: hhj_hi@sina.com. **罗 莉**,女,1971 年生,博士,副研究员,主要研究兴趣为体系结构、信息安全等. **董黎明**,男,1984 年生,硕士研究生,主要研究方向为信息安全. **何修雄**,男,1979 年生,硕士研究生,主要研究方向为信息安全. **侯方勇**,男,1971 年生,博士,副教授,主要研究兴趣为信息安全、体系结构等. **钟广军**,男,1974 年生,博士,副教授,主要研究兴趣为信息安全、图形学等.

蠕虫、木马的定义研究，情况要更糟糕些，有个别的形式化定义[4]，但影响力不大. 事实上，随着病毒技术的发展，很多病毒同时具备病毒、蠕虫、木马的特征. 例如，2001 年 9 月 8 日发现的 Nimda 病毒，NAI 公司把它归为病毒，CERT 把它归为蠕虫，而 Incidents. Org 则把它同时归为病毒和蠕虫. 从这个角度看，严格区分狭义病毒、蠕虫、木马的意义不大.

技术是中性的，本身不存在恶意，好人用它来造福社会，坏人用它来损人利己. 因此，如果定义了一种称为“病毒”的技术，并将采用该技术的程序称为病毒，那么，这种定义肯定是不严谨的. 据此观点，我们以是否违背用户意愿为标准，将广义病毒区分为显式病毒、隐式病毒，并给出了显式病毒的形式化定义和识别算法. 理论分析表明，狭义病毒以及大部分木马、蠕虫均属于显式病毒，实际案例分析也证实了这一点. 这些工作的重要价值在于，指出了狭义病毒以及大部分木马、蠕虫是可以准确识别的. 这一结论与“不能准确识别狭义病毒”历史结论并不矛盾，因为狭义病毒的经典定义实质上是定义了一种“病毒”技术，而不是病毒本身.

本文第 2 节提出程序是否为病毒是相对的，若对用户有损害则为病毒，否则不是，并简要分析传统病毒定义的不足；第 3 节提出“用户意愿”概念，用以描述用户操作计算机过程中的意图；第 4 节给出广义病毒的定义，并讨论定义的性质；第 5 节给出一种准确识别显式病毒的算法，并指出与传统结论截然不同的原因在于对病毒的定义不同；最后，总结全文.

2　病毒本质的讨论

2.1　损害的相对性

一个程序之所以被称为“病毒”，是因为它具有损害性. 损害性是恶意程序的本质，而是否造成损害一定是对用户而言的. 例如病毒 V，对于用户来说，如果他是在测试 V，那 V 对他而言是没有损害的，因为 V 运行的结果就是他所需要的；而如果他在不知道的情况下运行了 V 或者被 V 感染的程序，则 V 对他而言是有损害的，因为运行结果不是他希望的. 再例如，用户已经知道病毒 V 的功能是删除当前目录下的所有文件，用户经常把它当作一个文件删除工具使用，对用户来说 V 是一个实用程序.

可见，程序是否为病毒是用户的主观判断，这种认识可表述为以下公理.

病毒相对性公理. 一个程序是否为病毒，是相对于用户而言的. 如果用户认为程序给自己带来了损害就是病毒，否则不是病毒.

根据病毒相对性公理，我们可以直接得到一个非常有趣的结论：判断某程序是否为病毒的算法，如果不引入用户的意见，则该算法是不可能做出准确判断的. 这种认识可表述为以下定理.

判不准定理. 通过分析程序代码的方法准确判断一个程序是否为病毒，这样的判断算法是不存在的.

2.2　传统病毒定义的不足

为加深对病毒相对性的理解，我们回顾一下经典的病毒定义.

Cohen 1984 年给出了广为接受的病毒定义，“计算机病毒是一种程序，它可以感染其它程序，感染的方式为在被感染程序中加入计算机病毒的一个副本，这个副本可能是在原病毒基础上演变过来的”[2]. 随后，Cohen 给出了病毒的形式化定义[5]，将病毒的范围扩大到了所有的自我复制程序以及在系统或网络中演化和移动的程序.

基于上述狭义病毒的定义，有两个重要结论：(1) 很多正常程序被归结为病毒，从而病毒有了“好”、“坏”之分[6-7]. 例如自动升级程序，虽然不是病毒，但是按照 Cohen 给出的定义它们被归结为病毒. (2) 按照给定的病毒定义，没有算法能够准确识别病毒[1-2,4].

Cohen 的病毒定义没有引入用户的意见，因此，根据判不准定理有结论：不存在一个算法能够准确识别病毒.

事实确实如此，下面引用的是 Cohen 关于不能检测病毒的证明过程[2].

“为断定一个给定程序 P 是病毒，必须断定 P 感染其他程序. 这是不可判定的，因为 P 能够调用判断过程 D 并且感染其他程序，当且仅当 D 断定 P 不是病毒. 我们由此断定一个程序通过检查一个病毒的外表来区分病毒与其他程序是不可行的. 下面是对程序 V 的一个修改，作为 D 的不可判定性例子，我们使用了一个假定的判断过程 D，D 返回 true 当且仅当其参数是一个病毒.”

```
program contradictory-virus :=
{  ...
   main-program :=
   {  if ~D(contradictory-virus) then
      {  infect-executable;
         if trigger-pulled then do-damage;
```

```
        }
        goto next;
    }
}
```

记上述代码为 CV. 如果 D 断定 CV 是病毒，CV 将不感染其他程序，所以不是病毒. 如果 D 断定 CV 不是病毒，CV 将感染其他程序，所以是病毒. 因此，假定的过程 D 是自相矛盾的，通过外表来准确断定病毒是不可判定的."

3 用户意愿

为准确描述用户的意见，这里提出"用户意愿"概念. 用户意愿代表着用户的想法，理论上我们可以根据实际需要，定义各种各样的用户意愿. 本文仅讨论用户对程序访问文件的意愿.

用户意愿是用户发布的文件访问授权，表示同意某程序以某模式访问某文件. 例如，用户操作程序 Word，通过打开文件对话框选择了只读文件 doc，那么，用户意愿就是允许程序 Word 以只读模式访问文件 doc.

用户意愿强调授权信息的真实性，即这种访问授权确实是用户本人发布的，而不是程序伪造的.

定义 1. 一个用户意愿是用户发布的一个文件访问授权，表示授权程序 p 以模式 m 访问文件 f，记为(p,f,m).

用户授予程序某种访问权，程序可以长期性拥有这种访问权，也可以只是临时性的获得. 对应的，将用户意愿区分为静态意愿、动态意愿.

定义 2. 如果用户授权程序 p 以模式 m 访问文件 f，其有效时间是永久性的，那么，用户意愿(p,f,m)称为静态意愿.

例如，用户希望系统启动后自动运行某日程安排程序 MySchedule，那么其意愿"授权操作系统只读访问文件 MySchedule"就是一种静态意愿. 除非用户后来改变了这种授权，否则，每次系统启动时都会自动运行 MySchedule.

定义 3. 如果用户授权程序 p 以模式 m 访问文件 f，其有效时间是直到文件关闭，那么，用户意愿(p,f,m)称为动态意愿.

用户与程序的交互过程中，其发布的文件访问授权通常是动态意愿. 例如，用户操作程序 Word 编辑文档 doc，用户希望的是当他授权 Word 打开 doc 时，Word 才能访问 doc，关闭 doc 后，Word 不能访问 doc.

定义 4. 用 I 表示用户意愿的集合，用 F_{all} 表示计算机上所有文件的集合. 称 $F_{in}=\{f\mid\exists p\,\exists m(p,f,m)\in I\}$为意愿内文件集合，$F_{out}=F_{all}-F_{in}$为意愿外文件集合.

计算机运行过程中，用户意愿的集合随时间在变化，相应地，F_{in}、F_{out} 也在变化. 根据经验，$|F_{out}|>10000$，$|F_{in}|<10$，并且 F_{in} 随时间变化的幅度很小. 特殊情况下，例如扫描文件系统，F_{in} 可能接近或者等于 F_{all}.

读者或许从字面上理解"用户意愿"，认为意愿是一种主观想法，具有强烈的不确定性：对不同的用户来说，同一件事情可能有不同的意愿，甚至相反的意愿. 下面作进一步澄清. 用户意愿是由用户进行的工作明确决定的. 例如，用户运行程序 Word 编辑文件 f1，这决定了一个动态意愿(Word，f1，读写)，即授权 Word 以读写模式访问文件 f1；随后，用户关闭了 f1，这决定了取消一个动态意愿(Word，f1，读写)；再后来，用户编辑文件 f2，这决定了一个动态意愿(Word，f2，读写). 静态意愿也是明确的，由用户的需要唯一决定，前面解释静态意愿的定义时已经举过例子. 这可表述为以下性质.

性质. 用户意愿是明确的，动态意愿由用户进行的工作唯一决定，静态意愿由用户的需要唯一决定.

4 病毒的新定义及性质

4.1 病毒的新定义

有了"用户意愿"概念，就可以准确描述用户的意图，而从代码的执行行为是否符合用户意愿角度则可以定义损害性. 因此，可以以是否遵循用户意愿作为病毒判断标准.

定义以下记号：

p,p'：任意的程序.

m,m'：任意的文件访问模式.

$ACCESS_p$：$ACCESS_p=\{(f,m)\mid$ 程序 p 以模式 m 访问了文件 $f\}$.

定义 5. 一个程序 p 是显式病毒(简称 EV)，当且仅当，$\exists f\,\exists m((f,m)\in ACCESS_p\wedge(p,f,m)\notin I)$.

4.2 性 质

根据 EV 的定义，有以下覆盖能力结论.

定理 1. EV 涵盖了所有攻击 F_{out} 中文件的恶意程序以及攻击 F_{in} 中文件的两类恶意程序：(1) 不

按照用户授权的模式进行访问；(2) 访问其它程序被授权访问的文件.

证明.

先证明第 1 条结论：EV 涵盖了所有攻击 F_{out} 中文件的恶意程序.

$\forall p$，假设 p 攻击了 F_{out} 中的文件.

$\rightarrow \exists f \exists m((f,m)\in ACCESS_p \wedge f\in F_{out})$

$\rightarrow \exists f \exists m((f,m)\in ACCESS_p \wedge f\notin F_{in})$，

又 $F_{in}=\{f \mid \exists p \exists m(p,f,m)\in I\}$，

$\rightarrow \exists f \exists m((f,m)\in ACCESS_p \wedge (p,f,m)\notin I)$

$\rightarrow p$ 是 EV.

下面证明第 2 条结论.

(1) $\forall p$，假设 p 不按照用户授权的模式访问 F_{in} 中的文件.

$\rightarrow \exists f \exists m \exists m'((f,m)\in ACCESS_p \wedge (p,f,m)\notin I \wedge (p,f,m')\in I \wedge m\neq m')$

$\rightarrow \exists f \exists m((f,m)\in ACCESS_p \wedge (p,f,m)\notin I)$

所以 p 是 EV.

(2) $\forall p$，假设 p 访问程序 p' 被授权访问的文件.

$\rightarrow \exists f \exists p' \exists m((f,m)\in ACCESS_p \wedge (p,f,m)\notin I \wedge (p',f,m)\in I \wedge p\neq p')$

$\rightarrow \exists f \exists m((f,m)\in ACCESS_p \wedge (p,f,m)\notin I)$

$\rightarrow p$ 是 EV.

证毕.

根据传统病毒的定义，病毒的感染环节一定要首先分析程序文件，发现适合感染后一定要修改该程序文件. 由于程序文件是稳定的对象，用户不会授权程序对它进行修改，狭义病毒的这种读/写文件访问不属于用户授权范围. 因此，狭义病毒属于 EV. 同样道理，蠕虫、木马为了能够在计算机上永久驻留，必须将自身以单独的文件形式或者寄生在其它文件中的形式保存在系统中. 大部分的蠕虫、木马是这样的，也属于 EV. 这可表述为以下结论.

定理 2.　EV 涵盖了狭义病毒以及涉及文件篡改、文件窃取的蠕虫、木马.

为验证上述结论，我们随机抽取了 Trend Lab 提供的实际病毒案例共 255 个进行分析. 结果表明，这些病毒在执行过程中都包含了对文件系统的访问操作，特别是大多具有安装环节，在安装环节中修改注册表以保证在系统启动时能够自动运行. 这种文件系统访问操作，是违背用户意愿偷偷进行的，因而都是 EV.

定理 3.　EV 扩大了传统广义病毒的内涵，涵盖了部分有设计或开发缺陷的程序，如果这种缺陷导致程序进行非授权文件访问.

上述结论可由 EV 的定义直接得到. 这种范围扩大的重要意义，就在于把这类有缺陷的程序纳入到了需要防御的范畴，更好地满足了用户的需求. 考虑一个例子，为保证机器丢失时信息不泄密，设计这样一个自毁程序：当连续 10 次输入登录密码错误时，删除系统中的所有信息. 由于程序设计中存在有缺陷，程序在非预定条件下激活了自毁功能，毁掉了系统的信息，造成了损失. 这样的程序是不是病毒是有争议的，但是需要防御它却是没有争议的.

为覆盖所有的病毒，我们将 EV 之外的病毒定义为隐式病毒(简称 HV).

定义 6.　一个病毒 v 是 HV，当且仅当，v 不是 EV.

从文件系统角度看，HV 能够带来的后果是有限的，原因在于：(1) 只要程序 p 的攻击对象超出 F_{in}，p 就属于 EV. (2) 相比整个文件系统，F_{in} 通常非常小，HV 能够造成的损失也就限定在很小的范围内. 参见图 1，小方格区域为 F_{out}，小圆圈区域为 F_{in}.

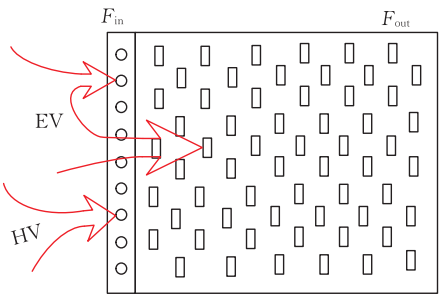

图 1　EV、HV 攻击范围示意

4.3　案例分析

为证实大部分病毒为 EV，我们在 2004、2006 年对 Trend Lab 提供的实际病毒案例进行了 3 次随机分析，共分析了 255 个病毒. 分析结果表明，这 255 个病毒全部为 EV，其中传统病毒 45 个，蠕虫 178 个，木马 32 个. 2009 年，我们又对 1998 年以来最具影响力、破坏力的典型病毒进行了分析，共分析了 19 个病毒. 分析结果表明，17 个病毒属于 EV，有对文件系统的读写操作和对注册表的修改操作，而 Code Red、SQL Slammer 不属于 EV. 分析结果见表 1，考虑到篇幅限制，仅给出个别病毒的原理分析.

图 2 给出了著名的“熊猫烧香”病毒的感染、传播流程. 该病毒于 2006 年 10 月 16 日爆发，并迅速登上年度病毒之王的宝座，被评为全球病毒史上最

具有影响力的病毒之一.从流程图可以看出,病毒运行会对磁盘文件系统进行访问操作(图中以灰色背景显示),且具有安装和修改注册表的环节.

表 1 著名病毒的主要原理分析

序号	病毒名	病毒类型	主要原理分析
1	CIH	文件病毒	利用 VXD 技术,直接攻击、破坏 PE 文件(EXE、DLL 文件).
2	WIN32.FunLove.4099	文件病毒	
3	Melissa	宏病毒	
4	I LOVE YOU	蠕虫	
5	Red Code	蠕虫	
6	SQL Slammer	蠕虫	利用 SQL SERVER 2000 的解析端口 1434 的缓冲区溢出漏洞,对网络上的 SQL 服务器进行攻击,该病毒也只存在于内存中.
7	Blaster	蠕虫	
8	Sobig.F	蠕虫	
9	Bagle	蠕虫	
10	My Doom	蠕虫	
11	Sasser	蠕虫	利用 LSASS 的缓冲区溢出漏洞进行攻击,并感染计算机上的 PE 文件.病毒运行后,将自身复制为%WinDir%\avserve.exe,并在注册表的相关启动项中创建:"avserve.exe"=%WinDir%\avserve.exe,以实现自动启动.
12	HappyTime	蠕虫	
13	Nimda	蠕虫	
14	Want.Job	蠕虫	
15	WhBoy.h	蠕虫	
16	Trojan/QQPass	木马	
17	win32.hack.huigezi	木马	
18	Trojan/PSW.GamePass	木马	
19	Trojan/Agent.pgz	木马	

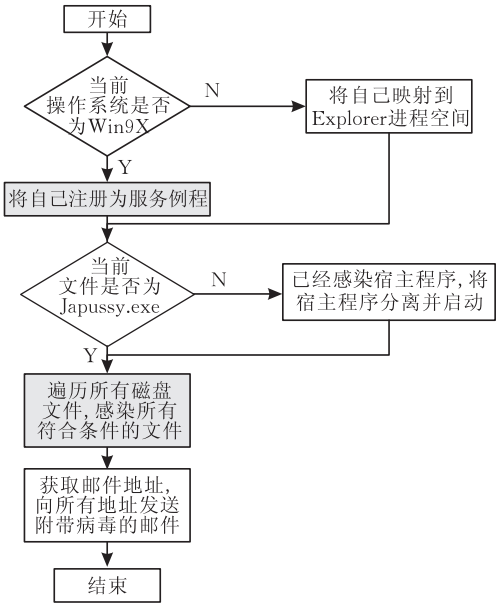

图 2 熊猫烧香病毒的感染、传播流程

综合以上案例分析,可以得出:绝大多数病毒在藏匿、感染、传播和破坏等过程中,都存在着对文件的读写和对注册表的修改.其原因在于:病毒为在计算机上永久保留下来,必须把自身的拷贝以文件形式保存起来;病毒为达到自动启动的目的,必须改写注册表文件.

上面案例中的"Red Code"和"SQL Slammer"病毒属于例外,原因是:它们都是利用缓冲区溢出漏洞,借助于服务器的网络连接(通过某些开着的端口对外大量地发送自己)来攻击其它的服务器,将病毒自身直接从一台电脑内存传播到另一台电脑内存中,它们并不往被攻击服务器的硬盘中写入病毒信息.它们攻击的对象并非终端计算机用户,而是网络服务器.

5 病毒识别算法及讨论

5.1 算法定义

病毒识别算法. 设 U_A 是当前登录系统的用户,p 为任意程序,I 是用户意愿的集合,I_S 是用户的静态意愿,(p,f,m) 表示"用户授权了程序 p 以模式 m 访问文件 f",$op(p,f,m)$ 表示"程序 p 请求以模式 m 访问文件 f",将目录当作文件看待.算法步骤如下:

初始化 I_S;
while(true) do
{ 等待事件 E 的发生;
if ($E=U_A$ 动态授权 p 以模式 m 打开文件 f) then
{ $I=I\cup\{(p,f,m)\}$; break; }
if ($E=U_A$ 静态授权 p 以模式 m 打开文件 f) then
{ $I_S=I_S\cup\{(p,f,m)\}$; 将 I_S 以文件形式保存在系统中; break; }
if ($E=p$ 退出) then
{ 从 I 中删除所有的(p,f,m); break; }
if ($E=p$ 关闭了文件 f && $(p,f,m)\in I$) then
{ $I=I-\{(p,f,m)$; break; }
if ($E=U_A$ 取消静态授权(p,f,m)) then
{ $I_S=I_S-\{(p,f,m)\}$; 将 I_S 以文件形式保存在系统中; break; }
if ($E=U_A$ 取消动态授权(p,f,m)) then
{ $I=I-\{(p,f,m)\}$; break; }
if ($E=p$ 发出请求 $op(p,f,m)$) then
{ **if ($op(p,f,m)$ 是"读目录名称、属性信息或目录下的文件名称、属性信息") then**
{ **执行 $op(p,f,m)$; break;** }
if ($(p,f,m)\in I\cup I_S$) then
{ **执行 $op(p,f,m)$; break;** }

```
        报警"p 是 EV";
    }
}
```

注意算法中标记为黑体字的语句块，它监控了所有的文件访问请求 $op(p,f,m)$，如果 $(p,f,m)\notin I\cup I_S$ 就被认为是病毒. 而这正好是 EV 的判断条件. 因此，直接有以下结论.

识别能力性质. 如果程序 p 是 EV，那么 p 一定会被病毒识别算法检测出来.

5.2　识别能力的讨论

识别能力性质表明，包括传统病毒在内的显式病毒可以被准确检测出来. 这与"病毒不可准确检测"的著名论断相矛盾. 这是否意味着前人的工作存在错误呢？

其实不然，原因就在于对病毒的定义完全不同. 传统的病毒定义，实质上是定义了一种称为"病毒"的技术，这种定义肯定是不完善的. 原因在于：首先，技术是中性的，本身不存在善恶，有人用它来造福社会，有人用它来损人利己；其次，病毒程序设计技术是发展的.

对此，Cohen 博士也意识到了，认为"就像任意的新技术一样，病毒是一把双刃剑"[6]. 例如，钩子函数技术被许多恶意软件用来窃取用户账户、密码等，根据狭义病毒定义，使用该技术的程序可以归结为病毒，但是许多正常软件采用该技术使自身功能强大，使用户使用软件更方便.

6　未来工作

应用广义病毒定义进行病毒防御，如果频繁向用户询问其意愿，会造成用户反感，甚至迫使用户放弃该方案. 因此，寻求全面准确的自动获取用户意愿的方案是下一步研究的重点.

此外，广义病毒定义虽然在安全性质、识别能力上比传统的病毒定义有较大突破，但仍然存在不足. 比如，隐式病毒没有给出准确、具体的形式化定义，这也是下一步的努力方向.

7　总　结

本文以是否违背用户意愿为标准，将广义病毒区分为显式病毒、隐式病毒，并给出了显式病毒的形式化定义和识别算法. 理论分析表明，狭义病毒以及大部分木马、蠕虫均属于显式病毒，实际案例分析也证实了这一点. 本文工作的意义在于从用户意愿角度给出了显式病毒、隐式病毒的定义，指出传统病毒以及大部分木马、蠕虫在理论上是可以准确识别的.

参　考　文　献

[1] Singh Prabhat K, Lakhotia Arun. Analysis and detection of computer viruses and worms: An annotated bibliography. ACM SIGNPLAN Notices, 2002, 37(2): 29-35

[2] Cohen F. Computer viruses-theory and experiments. Computers and Security, 1987, 6(1): 22-35

[3] Chess David M, White Steve R. An undetectable computer virus//Proceedings of the Virus Bulletin Conference. Orlando, FL, 2000

[4] Adleman L M. An abstract theory of computer viruses//Goldwasser J ed. Advances in Cryptology. LNCS 403. New York: Springer-Verlag, 1988: 354-374

[5] Cohen F. Computational aspects of computer viruses. Computers and Security, 1989, 8(4): 325-344

[6] Fred Cohen. A Case for Benevolent Viruses. California, USA: ASP Press, 1991

[7] Bontchev Vesselin. Are 'Good' computer viruses still a bad idea ?//Proceedings of the EICAR'94 Conference. Saint Albans, Hertfordshire, UK, 1994: 25-47

HE Hong-Jun, born in 1968, Ph. D., associate professor. His research interests include information security and software engineering economics.

LUO Li, born in 1971, Ph. D., associate professor. Her research interests include information security and computer architecture.

DONG Li-Ming, born in 1984, M. S. candidate. His research interests include information security.

HE Xiu-Xiong, born in 1979, M. S. candidate. His research interests focus on information security.

HOU Fang-Yong, born in 1971, Ph. D., associate professor. His research interests include information security and computer architecture.

ZHONG Guang-Jun, born in 1974, Ph. D., associate professor. His research interests include information security and computer graphics.

Background

In security domain, the definition of malicious software is a hot. Presently, there is only formal definition of virus, but no widely accepted definitions of worm and Trojan horse. However, according to traditional definition of virus, there is no algorithm can identify virus definitely. Definition of virus is the base to develop defense method, so it's valuable to work on this problem.

The paper breaks through traditional viewpoint on malicious software, proposes that whether a program code is a virus is relative to user, and only those bringing damage to user are viruses. Further more, the paper distinguishes viruses to explicit virus and hidden virus based on user's intention, and presents a formal definition of explicit virus and its identifying algorithm. Analyzing theoretically, traditional virus, most of worm and Trojan horse are explicit viruses, because that virus would insert itself to a file in order to retain on the machine, and virus would access registry file in order to run automatically. The authors have analyzed more than 270 actual viruses, the results show that traditional virus, most of worm and Trojan horse are explicit viruses, which is consistent with the theoretical analysis.

The significance of this paper is that a new formal definition of virus is proposed from viewpoint of user's intention, and points out that most of viruses can be theoretically identified definitely.

The work is supported by the National High Technology Research and Development Program (863 Program) under Grant No. 2009AA012428, which aims to develop a new method to defend unknown virus. No doubt, definition of virus is an important base for this project.