第33卷 第20期 计 算 机 工 程 2007年10月
Vol.33 No.20 Computer Engineering October 2007
·安全技术· 文章编号：1000—3428(2007)20—0181—03 文献标识码：A 中图分类号：TP393

基于系统调用挂钩的隐蔽木马程序检测方法

梁 晓，李毅超，崔 甲，曹 跃
(电子科技大学计算机科学与工程学院网络攻防实验室，成都 610054)

摘 要：隐蔽木马程序的设计本质是劫持常规的执行路径流，当前大多数检测手段无法全面检测出隐蔽性日益增强的木马程序。该文结合操作系统程序执行流程的局部相关性与确定性，在分析用户进程空间与内核空间中系统函数调用标志信息的基础上，检测系统中是否存在木马程序设置的隐蔽性系统调用挂钩，设计并实现了相应的检测方法。与现有的检测方法相比，该方案弥补了检测未知木马的不足，检测结果更全面。

关键词：特洛伊木马；RootKit；系统调用；挂钩；入侵检测；

Stealthy Trojan Horse Detection Method Based on System Call Hook

LIANG Xiao, LI Yi-chao, CUI Jia, CAO Yue
(Laboratory of Network Attack & Defense, School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu 610054)

【Abstract】 Trojan horses design essence lies in hijacking execution routine, and most of current detection methods fail to completely identify such ever-increasingly covert Trojan horses. The paper presents an approach to detect the existence of system call hooks set by Trojan horses based on the locality and determinacy of execution flows, and the analysis of system function call labs in both user and kernel levels, then designs and realizes corresponding prototype. Compared with current detections, the method offsets the deficiency in identifying unknown Trojan horses with more complete detection results.

【Key words】 Trojan horse; Rootkit; system call; hook; intrusion detection

现有的反病毒软件和木马检测软件，可以检测出一些常规的计算机木马程序，但随着网络入侵技术的发展，新一代隐蔽控制技术 Rootkit 的出现，使得攻击者能够获取对系统的完全访问控制权限，而更重要是，它隐藏于计算机系统之中，不会被管理员和安全检测软件所发现。现有的安全检测方法和工具由于受 RootKit 程序的欺骗攻击，无法检测出存在于系统中而实行了隐蔽功能的特洛伊木马和其他恶意程序。因此，检测秘密隐藏在系统中的特洛伊木马程序已成为网络安全方面的重要课题。

1 相关工作

恶意程序检测的一个关键问题是要能够发现未知的、隐蔽的恶意程序。下面简单介绍一些具有代表性的木马程序检测方法。

(1)完整性检测[1]。通过检查文件当前内容计算的校验和与原来保存的校验和是否一致，来发现文件是否有所变动。这种方法能发现已知和未知的恶意程序，但却不能进一步识别恶意程序类型。由于文件信息的改变有可能是正常程序引起的，因此完整性检测方法有较大的误报率。此外，完整性检测法在隐蔽性恶意程序Rootkit面前仍然无效。因为Rootkit在系统中一旦安装成功之后，它就能对完整性检测工具进行欺骗，使其得到与原始保存校验和相同的虚假的现有校验和信息。

(2)特征码扫描[2]。主要包括特征代码扫描法、特征字扫描法。特征码扫描检测技术中的特征码包括很多不同的属性，该方法针对已知恶意程序具有很高的准确率，误报率低。但是这种方法不能用来检测未知恶意程序，并且当系统中安装了Rootkit后，常规的特征码扫描反病毒软件将无法发现Rootkit在系统中的存在[3]。

(3)机器学习方法。主要是从大量的、不完全的、随机的数据集中寻找潜在的有价值的信息。因为一个新的恶意程序可能不包括已知的特征，为此提出了多种机器学习方法用于恶意程序的检测。RIPPER方法[4]是一种归纳学习方法，产生基于特征属性的布尔规则用于检测今后的恶意程序。多贝叶斯分类器[5]通过一组简单贝叶斯分类器的投票来确定一个实例的最终分类结果。以上方法都需要大量或是完备的程序组成的数据集，并将这些程序分成恶意和普通两种类型，才能达到比较高的检测性能，并且机器学习的训练时间较长。

2 基于系统调用挂钩的检测

木马程序的两个最基本的特点就是为入侵者提供长久的对被侵占主机的访问权限，以及实施非常隐蔽的入侵行为。为了达到隐蔽的要求，木马程序必须修改操作系统的执行路径，或直接修改操作系统保存的关于进程、驱动、网络连接等信息。系统调用挂钩，其目的在于劫持常规的执行路径流，然后修改由操作系统报告函数返回的信息，以迷惑和欺骗安

基金项目：国家科技基础条件平台工作基金资助项目(2003DIA7J051)
作者简介：梁 晓(1983－)，女，硕士研究生，主研方向：计算机系统安全，网络安全；李毅超，副教授；崔 甲，硕士研究生；曹 跃，助教
收稿日期：2006-12-11 **E-mail：**yuecao@uestc.edu.cn

全检测工具，从而达到隐蔽自身的目的。

另一类基于内存空间的检测方法是查询操作系统和系统进程空间中存在的挂钩。在系统的很多部分都可以隐藏一个系统调用挂钩，包括如下方法：导入地址表挂钩(IAT Hook)，系统服务描述符表挂钩(SSDT Hook)，中断描述符表挂钩(IDT Hook)，驱动输入输出请求报文处理挂钩(IRP Hook)，内嵌函数挂钩(inline Function Hook)。

2.1 导入地址表挂钩检测

最简单的用户态挂钩就是导入地址表挂钩。在应用程序使用一个来自其他二进制文件提供的函数时，应用程序必须导入这个函数的地址。几乎所有的应用程序使用 Win32 API 都是通过导入地址表来实现的。应用程序使用的每个动态链接库都包含在应用程序在文件系统中的一个映像中，这个结构称之为 IMAGE_IMPORT_DESCRIPTOR。它包括了动态链接库的名称和两个指向 IMAGE_IMPORT_BY_NAME 结构数组的指针。结构 IMAGE_IMPORT_BY_NAME 包含了应用程序需要的导入函数的名称。在应用程序被操作系统装载到内存空间中后，它所需要的动态链接库一旦被映射到内存空间，操作系统将定位每个需要的导入函数，并重写结构 IMAGE_IMPORT_BY_NAME 中对应函数的真实地址。

当木马程序的挂钩函数加载到应用程序地址空间中后，木马程序就能够分析目标应用程序的 PE 文件格式，根据需要将挂钩函数在导入地址表中的地址替换为笔者的挂钩函数的地址。当目标函数被调用时，木马程序的挂钩函数将会代替原始的目标函数执行。一旦执行完木马程序的挂钩函数，程序流将返回到原始目标函数的代码执行位置。

最基本的检测系统调用挂钩的算法是寻找执行到可接受范围外的分支路径。这些分支可能是由 call 或 jmp 之类的指令产生的。定义一个可接受的地址范围，在大多数情况下是比较容易的。在进程中，导入地址表显示列出包含导入函数的模块名称。这些模块在内存中有一个明确的开始地址和一个长度。根据这些地址，可以计算出可接受地址范围。

2.2 系统服务描述符表挂钩检测

在内核结构中，Windows 内部系统服务的地址被保存在称为系统服务调度表的结构中。在内存中，可以通过系统调度索引号定位在本调度表中函数地址。另一个表为系统服务参数表，它被用来指定每个系统服务函数中参数的数目和长度。在 Windows 内核中导出了一个 KeServiceDescriptorTable 表结构，它包含了一个指向系统服务调度表的指针，指向在内核中提供的核心系统服务的实现。KeServiceDescriptorTable 同样也包含了一个指向系统服务参数表结构的指针。为了调用一个指定的函数，系统服务调度器将会获取一个指定服务的 ID 号，通过它计算出在系统服务调度表中的偏移量以定位相应的函数指针。同时，在 KeServiceDescriptorTable 中包含了系统能提供的最大服务数量值。

如果木马程序以设备驱动的方式加载到操作系统中，它就可以修改系统服务调度表的指针，使其指向一个自定义的函数，来代替操作系统提供的原始系统服务。当用户态的函数调用进入内核，这个请求被系统服务调度器处理，最终进入到笔者的木马程序提供的功能函数。

同样，对设备驱动来说，所有的合法的输入输出请求报文处理函数都应该存在于一个给定的驱动地址范围之内，而系统服务分配表中的所有入口地址都应该在内核进程 ntoskrnl.exe 的地址空间内。

2.3 中断描述符表挂钩检测

中断描述符表是用来处理系统中断的，这些中断可以来自软件或硬件设备。在 Windows 系统中，用户态进程请求系统服务描述符表的访问时，会发出 0x2E 号中断。其中，笔者最关心的是在中断描述符表中为 0x2E 号中断安装一个挂钩，该挂钩函数将在所有 SSDT 中的内核函数之前被执行。

需要注意的是，执行控制流程是不会返回到 IDT 处理函数中的，因此常规的挂钩技术(如调用环视函数，过滤数据，从挂钩函数返回到原始函数)这里不能通用。中断描述符表挂钩是一个经过式函数，它不会获取控制权，所以不能用来修改或过滤数据。但是，木马程序用它来鉴别或阻塞来自特殊进程的请求，比如主机入侵防御系统或个人软件防火墙。

查询中断描述符表挂钩更困难，因为不知道绝大多数中断的可接受地址范围。但明确知道的一个中断地址就是 INT 2E 中断处理函数的地址，指向内核空间 ntoskrnl.exe 之中。

2.4 输入输出请求报文处理挂钩检测

另外一个在内核中能实现很好隐藏的地方是每个设备驱动的函数表。在设备驱动被安装后，它将初始化处理各种不同输入输出请求报文的函数指针表。输入输出请求报文处理几类特殊的请求，比如读、写、查询等。因为设备驱动在控制流中处最底层，对木马程序来说，它们也是实施挂钩的理想地方。木马程序感兴趣的输入输出请求报文的类型与它们希望实现的功能有很大的相关性。比如，可以挂钩处理文件系统写或 TCP 请求的函数。但是，它同样有中断服务调度表挂钩类似的问题，那就是它不能调用原始的函数并过滤数据。在调用堆栈中，这些函数不能够从底层的设备驱动中返回。

输入输出请求报文挂钩的检测比较简单。在系统内存中查询出操作系统已经装载的所有驱动列表，然后可以在每个特别的驱动中，定位 IRP 处理表的位置。结合上面两种方法，就可以获取系统中存在的输入输出请求报文处理挂钩。

2.5 内嵌函数挂钩检测

第 2 个用户态挂钩方式为内嵌函数挂钩，它比导入地址空间挂钩更强大。导入地址表挂钩不能挂钩动态载入连接库中的函数，而内嵌函数挂钩不会出现这样的问题。在实施内嵌函数挂钩时，木马程序将会重写目标函数的代码字节，因此不管应用程序何时解析函数地址，它都将会被挂钩。内嵌函数挂钩技术可以应用在用户态函数和内核态函数。

一般来说，一个内嵌函数挂钩的实现是基于保存目标函数的起始几个字节，木马程序将会重写这几个字节的信息。在原始字节内容被保存后，它们往往会被一个跳转指令(如 jmp, call)所替代。这个跳转直指木马程序的挂钩函数。执行完木马程序相关指令后，通过保存的原始被替换字节的内容，程序流将重新返回执行目标函数。

内嵌函数挂钩的检测是最困难的，因为挂钩可以被定位在函数中的任何位置，而且函数可以调用在模块地址范围外的指令。

在进行系统调用挂钩扫描时，会面临内存扫描方法所遇到的全部问题：已经被加载到系统内存之中并正在执行的木马程序，可能会干扰检测方法。但是，使用系统调用挂钩的检测方式最大的好处在于其显著的通用性。通过对系统调用挂钩的检测，笔者将不会再面对寻找已知特征值或已知方式的难题。

3 实现与结果

针对本文提出的基于系统调用挂钩的隐蔽木马程序检测

技术，设计并实现了该检测方式相应工具。该工具针对当前木马程序自身隐藏的特性，通过分析操作系统用户进程空间和内核空间系统调用的标志信息，来分析系统中是否存在木马程序设置的系统调用挂钩，以检测出隐蔽的木马程序。该木马检测软件的模块设计如图 1 所示。

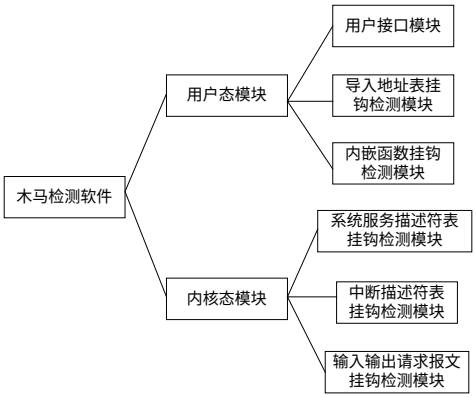

图 1　木马检测软件模块设计

该木马检测软件在 Windows 平台上用 VC 开发实现，最后生成用户态可执行文件和系统驱动文件。用户通过可执行文件中的选项，实现对不同类系统调用挂钩的检测。一旦需要对内核空间的系统调用挂钩进行检测时，用户态可执行程序将该系统驱动文件加载到内存，通过消息队列来实现对内核模块的调用和检测。

目前，该工具可以检测已公开的具有隐蔽特性的典型木马程序包括：Vanquish，用户态隐藏的木马程序；Hacker Defender，用户态与内核态相结合隐藏的木马程序；yyt，内核态隐藏的木马程序；NtRootKit，内核态隐藏的木马程序等。

4　结论

基于系统调用挂钩的隐蔽木马程序检测技术是针对现有的系统恶意程序自身隐藏行之有效的检测方法，但在实现过程中，同样会面对入侵者的攻击，使得通过本算法检测的结果数据被修改。同样，针对在系统中通过直接修改系统内核数据的方式来实现自身隐藏的计算机木马，该方法不能将其检测出来，这也是下一步重点研究的方向。所有的隐蔽木马程序都是由操作系统设计的不安全性造成的，如何加强操作系统的安全也是另一个重要的研究内容。

参考文献

1 Fiskiran A M, Lee R B. Runtime Execution Monitoring (REM) to Detect and Prevent Malicious Code Execution[C]//Proceedings of the IEEE International Conference on Computer Design. San Jose, CA. 2004.

2 Deng P S, Wang J H, Shieh W G, et al. Intelligent Automatic Malicious Code Signatures Extraction[C]//Proceedings of IEEE 37th Annual 2003 International Carnahan Conference on Security Technology, Taiwan, China. 2003.

3 Wang Yimin, Beck D, Binh V, et al. Detecting Stealth Software with Strider GhostBuster[C]//Proceedings of the 2005 International Conference on Dependable Systems and Networks. Yokohama, Japan. 2005.

4 Cohen W W. Learning Trees and Rules with Set-valued Features[C]//Proceedings of the 13th National Conference on Articial Intelligence, Portland, Oregon. 1996.

5 Schultz M G, Eskin E, Zadok E, et al. Data Mining Methods for Detection of New Malicious Executables[C]//Proceedings of IEEE Symposium on Security and Privacy, Oakland, California. 2001.

(上接第 177 页)

进一步，如果考虑$ps1$，$ps2$ 的值对新个体而言都只影响 1 个bit的值，可以通过假设a_0，a_1,…，a_7和$(z_{64}\oplus a_{64})$，$(z_{65}\oplus a_{65})$，…，$(z_{71}\oplus a_{71})$这 14 个变量的值，此时利用$Z_{i+1}=KK1_{i+1}\oplus KK2_{i+1}$可建立相应的线性表示式，其中最多含有 4 个错误的式子。在容许最多 4 个含错方程的情况下，利用上面保守估计的方法，密钥空间的规模将由 2^{128}降低到 $2^{16+(128-16)/2}=2^{72}$。

另外，也可以根据杂交算子的保距性，仅假设a_0, a_1, a_2, a_3和$(z_{64}\oplus a_{64})$, $(z_{65}\oplus a_{65})$, $(z_{66}\oplus a_{66})$, $(z_{67}\oplus a_{67})$ 这 8 个变量的值，说明对密钥空间的规模的影响。杂交算子的保距性质如下：

引理[5] 设二进制编码的两个个体X，Y，经过单点杂交后得到新个体为 $\varphi_1(X,Y),\varphi_2(X,Y)$，则

$$X\oplus Y=\varphi_1(X,Y)\oplus\varphi_2(X,Y)$$

其中，$\oplus$是逐位模 2 加运算。

利用杂交算子的保距性质，可以在不知道交叉点位置的情况下，建立 64 个方程，即把$KK1_{i+1}$分成两部分，把$KK2_{i+1}$分成两部分，这 4 部分对应位的逐位模 2 加运算的值等于把Z_{i+1}分成两部分并进行逐位模 2 加运算得到的值。由于变异算子的作用，在这 64 个方程中也是最多含有 4 个错误的式子。可以通过假设a_0，a_1，a_2，a_3和$(z_{64}\oplus a_{64})$，$(z_{65}\oplus a_{65})$，$(z_{66}\oplus a_{66})$，$(z_{67}\oplus a_{67})$这 8 个变量的值，在容许最多 4 个含错方程的情况下，根据密钥序列的生成方法，采用上面保守估计的思想，密钥空间的规模将由 2^{128}降低到 $2^{8+64}=2^{72}$。

3　结束语

通过上面的分析可知，尽管文献[3]所设计的密码体制的密钥长度为 256bit，其密钥空间的规模也为 2^{256}，但如果已知密钥输出序列的一个截断，其密钥空间的规模将下降为 2^{128}；如果已知密钥输出序列的连续两个截断，其密钥空间的规模将下降为 2^{84}，并且在容许最多 4 个含错方程的情况下，密钥空间的规模将降低为 2^{72}。如果已知密钥输出序列的更多截断，其密钥空间将下降得更多。由此可见，从选择明文攻击的角度来说，该密码体制的安全性是极其脆弱的。

参考文献

1 吴世忠, 祝世雄, 张文政, 等. 应用密码学[M]. 北京: 机械工业出版社, 2003.

2 张焕国, 冯秀涛, 覃中平, 等. 演化密码与 DES 的演化研究[J]. 计算机学报, 2003, 26(12): 1778-1784.

3 胡能发, 邓永发. 基于遗传算法的序列密码生成方法[J]. 计算机工程与设计, 2005, 26(8): 2190-2192.

4 李敏强, 寇纪松, 林　丹, 等. 遗传算法的基本理论与应用[M]. 北京：科学出版社, 2002.

5 张文修, 梁　怡. 遗传算法的数学基础[M]. 2 版. 西安: 西安交通大学出版社, 2003.