基于线程调度的进程隐藏检测技术研究

梁 晓 李毅超

(电子科技大学计算机科学与工程学院 成都 610054)

摘 要 基于线程调度的进程隐藏检测技术,利用操作系统进程的资源分配和调度机理,通过直接扫描系统内核中的活动线程来逆向检测实际存在的进程列表信息。该方法可以检测出当前常规安全检测工具不能发现的系统恶意程序的入侵行为。和现有的进程隐藏检测方法相比,该检测方式克服了各种缺陷,具有更加彻底可靠的安全机制,可以检测出当前所有类型的进程隐藏。

关键词 特洛伊木马,Rootkit,进程隐藏,线程调度,入侵检测

Research on Thread Dispatch Based Hidden Process Detection Technique

LIANG Xiao LI Yi-Chao

(College of Computer Science and Engineering, UEST of China, Chengdu 610054)

Abstract Thread dispatch based hidden processes detection technique makes use of the process's resource assignment and dispatch mechanism in operating system to scan active threads in system kernel for reverse detecting active processes list. This method can detect more Trojan horse's intrusions than general security detection software. Comparing with normal hidden process detection techniques, it has gotten over all of the limitations, and found all types of current hidden processes based on more reliable secure mechanism.

Keywords Trojan horse, Rootkit, Process hiding, Thread dispatch, Intrusion detection

1 引言

特洛伊木马[1]作为一种危害性大、隐蔽性高的远程控制工具,通常被反病毒软件作为病毒来处理。但随着网络入侵技术的发展,出现了新一代的后门控制技术 RootKit[2]。她通过修改操作系统代码,使得攻击者能够获取对系统的完全控制权,同时隐藏于计算机系统之中而不被管理员和安全检测软件所发现。现有的检测技术都是基于信任操作系统所提供的主机运行信息这个前提,但由于 RootKit 能够修改系统的这些关键数据,使得常规安全检测方法和工具被欺骗而变得不再可信。进程隐藏是系统管理员面临的最通常的安全威胁,必须考虑将秘密潜伏在系统中的特洛伊木马进程检测出来,以实现可信任的主机入侵检测与防御。

本文提出基于线程调度的进程隐藏检测技术,从操作系统进程的资源分配和调度机理出发,通过直接扫描系统内核中的活动线程来逆向检测实际存在的进程列表信息。和现有的进程隐藏检测方法相比,该检测方式克服了各种缺陷,具有更加彻底可靠的安全机制,可以检测出当前所有类型的进程隐藏。

2 系统进程隐藏

RootKit 定义中,最后一个关键点是隐藏攻击者在系统中各项行为的存在,RootKit 包括多种隐蔽攻击者在系统中存在的功能,这其中就包含允许攻击者隐蔽当前系统中的进程信息[3]。

在 Intel 系列的处理器中,存在四个特权等级,但在一般的通用操作系统中仅仅存在两个等级,因此 RootKit 也可以运行在两个不同的层次上,这取决于它替换或修改的目标操作系统中的软件或模块。我们按 RootKit 对操作系统所做修改的不同分为用户模式 Rootkit 和内核模式 Rootkit。它们都是通过修改系统返回的进程列表信息来欺骗上层安全检测工具的。

2.1 用户态进程隐藏

用户态进程隐藏的实现机制包括动态链接库注入(DLL Injection),应用程序接口挂钩(API Hook),远程线程注入(Remote Thread Injection)等方式等。

(1)动态链接库注入也就是文件替换的方式,新创建的动态链接库文件中一些关键函数被修改过,如 ZwQuerySystemInformation 函数。这样,在调用被修改后的动态链接库文件中相应函数时,执行了被系统恶意程序重新定义的程序指令,返回给用户的是被修改后的进程列表信息。

(2)应用程序接口挂钩和动态链接库注入非常类似,但不需要替换文件。注入的过程是通过系统函数 OpenProcess(),WriteProcessMemory()等来实现的。使用这类技术的典型木马就是 Hacker Defender。它们动态打开目标进程的地址空间,然后寻找需要修改的特殊函数,这类函数主要包括获取系统进程信息的函数。最后在目标进程的地址空间内动态地修改函数代码,以实现对原始函数运行的劫持,也就是系统调用挂钩技术。

(3)远程线程注入方式,这种技术不需要其他额外进程的创建。恶意系统程序选择一个合法的进程,再将新线程的可执行代码注入到这个目标进程的地址空间中去,然后在目标进程空间中执行这段被注入的代码以实现特殊的功能。这个被注入的线程就可以为恶意系统程序执行想要的功能指令,

梁 晓 硕士研究生,研究方向:计算机网络与网络安全。李毅超 副教授,主要研究方向:计算机网络与网络安全。

其实质是原恶意程序没有长期存在的进程。这种方式也存在一个灵活性方面的问题，尽管一些任务可以用线程注入的方式来实现，但它不能拥有一个运行时隐藏的控制台，在这个控制台下也不能够启动新的不同程序。

2.2 内核态进程隐藏

内核态进程隐藏的实现机制包括内核服务挂钩，内核数据修改等。

内核服务挂钩是一类典型的进程隐藏方式，它将挂钩一些知名的内核服务函数，并修改这些函数的返回结果以欺骗上层调用程序。其中有几种实现方式：

(1)系统服务调度表(System Service Dispatch Table)挂钩：这是最流行的方式，文[4,5]对此进行了分析。Windows操作系统中的所有上层函数，在系统的内核底层都有对应的系统服务函数，当进程枚举查询类函数对应的内核系统服务被修改后，那么用户态获取的进程列表信息也将是不可信任的。但是这种方式可以很简单地通过比较当前操作系统和"干净"操作系统的系统服务调度表中的内容来判断是否存在这类挂钩。

(2)中断调度表(Interrupt Dispatch Table)挂钩：和系统服务调度表挂钩很类似。在Windows操作系统中，所有的用户态函数接口都是通过一个特殊的软中断进入内核的，也就是内核模式。通过挂钩这个软中断，可以实现对系统所有函数调用的监视和修改。当然，基于中断调度表挂钩的进程隐藏方式和系统服务调度表挂钩一样，也有相同的缺点。

(3)原始内核代码修改：它不是通过修改代码的指针来实现隐藏功能，而是程序代码本身。在程序代码中，可以通过修改函数的入口地址来实现对函数调用的修改，也就是调用原始的函数名时执行的却是其他函数的代码，通常称之为挂钩。同样，在目标函数代码指令的开头，由于存在一些固定的汇编指令，通过修改这些指令并插入相应的跳转指令就可以实现代码流的改变。这种方式是在被修改函数的代码内部实现程序流程的改变。

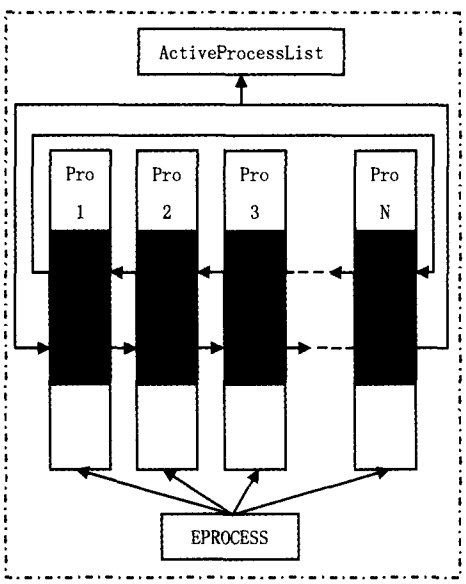

图1 活动进程列表结构

(4)特殊指针修改：通过修改一些非常规的代码指针(非IDT、SSDT)，来实现内核服务的执行路径流程。在操作系统内核中存在大量的这类指针，比如内核对象KTHREAD结构中的pServiceDescriptorTable指针就可以被恶意程序所利用，来达到进程隐藏的目的。

内核数据修改方法直接对系统的数据结构进行操作来隐藏系统中存在的特殊进程信息。在Windows操作系统中，存在一个保存当前系统中所有进程信息的双链表的内核数据结构如图1所示，这个结构的导出接口为PsActiveProcessLinkHead指针。

通过PsActiveProcessLinkHead指针可以枚举出系统中所有的存活进程列表。内核数据修改方法不是通过修改内核代码或执行路径，而是解开PsActiveProcessLinkHead列表中的进程链接，使得被隐藏进程脱离这个保存进程信息的内核数据结构。当内核态进程查询服务探测系统中存在的所有进程信息时，就是通过读取PsActiveProcessLinkHead结构中的进程信息，并返回给上层用户接口调用。在Windows操作系统中，因为PsActiveProcessLinkHead列表并不被系统进程调度代码所使用，因此解开链表后，操作系统内的线程仍然能够获取执行CPU的资源，但进程对象已经被隐藏起来了。

3 基于线程调度的进程隐藏检测技术

3.1 操作系统调度机制

在Windows操作系统中是基于线程对象的资源调度规则，因此在系统中存在不同状态的线程队列，如运行、等待状态等。恶意系统程序可以修改系统内核的活动进程列表来隐藏自身并获取CPU资源，但不能够修改系统内核的活动线程列表。如果恶意程序在系统的线程列表中解开了想要隐藏的线程的数据结构后，这些在队列中被解开的线程将不会再获取CPU地运行时间资源，这样恶意程序就不能在系统中正常地运行了。其本质在于Windows等操作系统是基于线程调度资源分配的，也就是操作系统不会给没有探测到的线程分配相应的CPU执行时间单元。

在操作系统中维持了三组系统调用的线程列表，它们是KiDispatcherReadyListHead，KiWaitInListHead和KiWaitOutListHead。其中第一个数据结构中包含了32个列表，每个列表对应一个调度优先级，全部列表构成了系统中存在的所有状态为READY的线程资源。基于分时优先权的调度机制，操作系统会在这32个列表中调度相应的线程，以各自分配应有的执行时间。后两个数据结构非常相似，它们维持了系统中处于等待状态的线程列表。鉴于操作系统中的CPU资源分配是基于线程来实施的，所以只要恶意系统程序存在并运行，就一定有相应的线程资源及其对应的进程信息。

3.2 基于线程调度的进程隐藏检测

在Windows操作系统中，可以通过系统的调试符号表来获取数据结构KiDispatcherReadyListHead，KiWaitInListHead和KiWaitOutListHead的地址信息，通过扫描这三个链表就能枚举出系统内部存在的所有的线程资源列表。基于每个资源对象，可以获取相对应的线程信息，而系统中存在的每个线程都有一个所属的进程资源信息，这样便可以检测出系统中真实存在的所有进程的列表信息。基于线程调度的进程隐藏检测算法如下：

```
1 for each element e on list KiWaitInListHead do
2    getProcessesFromElement(e);
3 for each element e on list KiWaitOutListHead do
4    getProcessesFromElement(e);
5 for each priority level in KiDispatcherReadyListHead do
6    for each element e on the i list do
7       getProcessesFromElement(e);
8 getProcessesFromWindowsUserLandAPI();
```

(下转第118页)

$P=|x_i-\bar{x}|/S$

在流量正常时，P 值较小，一般小于1；如果 P 的值超过2（这仅仅为经验值，根据实际情况的不同，P 值在2附近浮动），认为有异常流量发生或结束。

那么一个理想的小波函数在检测流量时，应该是：在异常流量发生或结束时应该有较大的偏离，而在其他情况下，偏离较小。

同时我们以100个数据为一个总体样本，即以100个采样点（100分钟）为一个分析检测窗口，为方便观察，统一将分析检测窗口的第50时间点作为异常流量的发生时间。

由于总体样本较小（100时间点），故只需进行三层分解，其中第1、2层看作高频部分，第3层看作中频部分，是我们最感兴趣的部分，也是包含奇异性信息最丰富的部分。我们以byte为参数，对小波分析的第三层细节系数用小波方差分析进行度量。采用四种小波函数：Db3、Db6、Coif3、Dmey，分别

表1 不同小波函数的效能

	均值 $\bar{x}$	异常流量发生点第三层系数 x	方差 S	$P=\|x_i-\bar{x}/S$
Db3	−366.3	7.11×10^4	3.67×10^4	1.95
Db6	−313.8	1.25×10^5	4.68×10^4	2、68
Coif3	362.0	1.95×10^5	5.15×10^4	3.78
Dmey	340.4	1.76×10^5	5.40×10^4	3.25

用这四种小波函数对SYN flooding异常流量进行检测分析，实验结果如表1，可以发现，Coif3具有最好的检测效果。我们还对其他异常流量进行了分析，发现coif3均具有较好的检测效果，限于篇幅，这里不作详细介绍。

结论与进一步的工作 本文提出了一种基于小波技术的网络异常流量检测方法，通过采集校园网流量数据，计算采样点的偏离值，用偏离值的大小来评估是否发生了流量异常，在实际应用中有着较为理想的效果。同时，进一步的研究也是必须的，重点将放在以下几个方面：不同的采样间隔对异常流量检测的影响；小波分解只对信号的低频段进行分解，而对高频段没有进一步的分解，那么使用小波包分解是否有着更为理想的效果；异常流量的分类研究，这将有助于我们对本文方法进行进一步的总结和提升；在开源平台上建立起一个自动检测异常流量的系统等。

参 考 文 献

1 Thottan M, Ji C. Proactive Anomaly Detection Using Distributed Intelligent Agents. IEEE Network, 1998, 12(5): 21～27

2 Alarcon-Aquio V, Barria J A. Anomaly detection in communication networks using wavelets. IEE Proceeding-Communication, 2001, 148(6): 355～362

3 Barford P, Kline J, Plonka D, et al. A signal analysis of network traffic anomalies. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop, 2002

4 Abry P, Veitch D. Wavelet analysis of long range dependent traffic. IEEE Transactions on Information Theory, 1998, 44(1)

5 http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html

6 http://www.splintered.net/sw/flow-tools/docs/

（上接第115页）

```
9 compareProcessesGetFromDifferentSource();
10 getHiddenProcessesList();
1  getProcessesFromElement (element e)
2     get the thread t which element e belongs to;
3     get the process p which thread t belongs to;
4     insert the process p in foundProcessList;
```

4 实现与结果

针对本文提出的基于线程调度的进程隐藏检测技术，可设计能检测出目前所有特洛伊木马的隐藏进程的工具软件。

OneTNS是本设计思想在Window平台下的进程检测工具，它针对当前计算机木马程序自身隐藏的特性，通过读取操作系统内核数据结构，来获取可靠的系统状态信息，实现可信任的主机入侵检测与防御。OneTNS目前可以检测已公开流行的具有进程隐藏功能的系统恶意程序包括：

Vanquish，用户态进程隐藏的恶意系统程序。

Hacker Defender，内核态进程隐藏的恶意系统程序。

Fu，基于直接修改内核对象的进程隐藏的恶意系统程序。

Dasher，2005年12月出现的基于多个Windows漏洞的蠕虫，包含了基于内核态系统调用挂钩的进程隐藏功能。

当以上系统恶意程序被安装于Windows操作系统中之后，通过现有的任务管理器等其他的进程检测监视工具不能发现它们的存在，而通过我们的检测软件将会完全探测出它们相关的进程信息，并成功实现对该进程的分析、结束、停止等控制操作。

结束语 基于线程调度的进程隐藏检测技术是针对现有的恶意程序中进程隐藏行之有效的检测方法，但在实现的过程中，同样会面对入侵者的攻击，使得我们通过本方法检测的结果数据被修改。因此，如何提高检测的完整性与可靠性，如何保护安全检测软件自身的行为等，是下一步重点研究的内容。另外，针对特洛伊木马程序的其他隐藏功能，利用操作系统的内核运行机理，也可以提出更进一步的检测方式思想。

参 考 文 献

1 Thimbleby H, Anderson S, Cairns P. A Framework for Modeling Trojans and Computer Virus Infections. The Computer Journal, 1998, 41(7): 444～458

2 Buteler J R II. Detecting Compromises of Core Subsystems and Kernel Functions in Windows NT/2000/XP: M.S. Thesis, University of Maryland, Baltimore County, 2002

3 Butler J, Jeffrey L. Undercoffer and John Pinkston. Hidden Processes: The Implication for Intrusion Detection. In: Proceedings of the 2003 IEEE Workshop on Information Assurance United States Military Academy, West Point, NY, June 2003

4 Levine J G, Grizzard J B, Hutto P W, Owen H L. A Methodology to Characterize Kernel Level Rootkit Exploits that Overwrite the System Call Table. In: Proceedings of IEEE. SoutheastCon, IEEE, 2004. 25～31

5 Levine J, Grizzard J, Owen H. A Methodology to Detect and Characterize Kernel Level Rootkit Exploits Involving Redirection of the System Call Table. In: Second IEEE International Information Assurance Workshop, 2004