

一种基于虚拟化哈佛体系结构的 Rootkit 技术

易再尧¹, 黄本雄¹, 孙建华²

(1 华中科技大学 电信系, 湖北 武汉 430074; 2 湖南大学 计算机与通信学院, 湖南 长沙)

摘 要: 通过修改操作系统内核的方式实现了哈佛内存体系结构的虚拟化, 在操作系统层分离了对 Rootkit 代码的取指和数据操作, 从而可以绕开内存扫描程序, 实现攻击代码的隐藏. 文中详细讨论了虚拟化哈佛体系结构的具体实现, 并分析了它的有效性和对操作系统各个方面的影响. 实验表明: 该方法能很好地隐藏攻击代码, 对操作系统的正常工作影响十分轻微.

关键词: 隐藏; 虚拟化; 内存扫描; Rootkit

中图分类号: TN452

文献标识码: A

文章编号: 1000-7180(2010)04-0129-04

A Rootkit Based on Virtualized Harvard Architectural

YI Zai-yao¹, HUANG Ben-xiong¹, SUN Jian-hua²

(1 Department of Electronics & Information Engineering, Huazhong University of Science & Technology, Wuhan 430074, China; 2 School of Computer & Comm., Hunan University, Changsha 410082, China)

Abstract: In this paper, the virtualization of Harvard memory architecture is implemented as a patch to the current operation system to separate the code fetch and the data operation of the rootkit code, and avoid being detecting by memory scanners to hide the attack code. The implementation of our virtualization is described in detail and its effectiveness to resist attacks and impact on the current operating system are fully analyzed. The experiment results show our approach is effective, has no side effect on the normal functions of the operation system.

Key words: hide; virtualization; memory scan; Rootkit

1 引言

计算机操作系统向下管理硬件资源, 向上为应用程序提供接口. 为了更好地利用底层硬件资源以及更好地为上层软件提供服务, 操作系统正变得日益复杂. 例如当前最流行的两个操作系统 Linux 和 Windows XP 的内核代码分别为 430 万行和 4000 万行. 如此纷繁复杂的操作系统内核代码必定存在一些漏洞, 通过利用操作系统的一些漏洞进行攻击已经成为黑客们的常用技术.

许多攻击技术需要实现十分复杂的功能, 而这些功能必然需要一定量的代码来实现. 这些功能代码必须很好地隐藏起来, 不被操作系统或者反病毒软件发现. Rootkit 技术正是这样一种隐藏技术. 经

过多年的发展, Rootkit 已经转向内核级别^[1], 开始通过直接修改内存中操作系统的内核对象来达到隐藏的目的^[2].

文中提出了一种新的 Rootkit 技术. 它利用虚拟化的方法在操作系统层实现哈佛体系结构的内存管理机制, 能够保证系统在被打上补丁之后, 对攻击代码所在内存区域的读操作都会重定向到物理内存中另外一个影子代码区域, 从而保证攻击代码的隐蔽性. 文中的方法充分利用了底层硬件的特性, 对操作系统的性能的影响不大.

2 Rootkit 技术

与传统的病毒或是木马不同, Rootkit 的目的是“隐藏”. 通过篡改系统上的文件或是某些敏感操作

收稿日期: 2009-03-30; 修回日期: 2009-07-05

基金项目: 国家“九七三”计划项目(2007CB310900)

系统对象,Rootkit 力图消除一切可能暴露自己的痕迹,以便长期潜伏,方便攻击者在未来某个时候重新控制系统. Rootkit 技术可以看作“中间人”攻击的一种,它一般通过拦截和改变操作系统的不同实体之间的通信接口来达到隐藏的目的^[3].

Rootkit 技术与系统保护、入侵检测技术一起不断发展. 在攻击者和保护在相互角力的过程中,Rootkit 技术不断演化,经历了从用户级到内核级,从简单的修改磁盘文件到直接操纵内核对象(Direct Kernel Object Manipulation)的过程,Rootkit 技术还将进一步发展,其隐藏能力将越来越强.

现在许多的内核 Rootkit 采用的方法是在内核的数据空间中注入自己的代码,然后通过某种方式使内核执行这部分代码. 但是,这并不代表内核级 Rootkit 有能力修改内核代码. 有些 Rootkit 就是通过修改操作系统某些系统调用的代码来达到隐藏的目的^[4]. 还有些攻击者甚至可以通过操纵内核的内存管理机制,将某些页面的线性地址对应的物理地址修改为自己指定的某个物理地址^[5],这种方法如果用在对外核代码的篡改上,将会带来很大的威胁.

由于许多使数据区代码不可执行技术的出现,代码注入型的攻击会越来越少,取而代之,更多的 Rootkit 可能会采用非注入型攻击的方法(例如 non-control-data 攻击^[6])或是直接篡改内核代码来放入自己的代码. 所以,将来可能会有更多的 Rootkit 使用篡改内核代码的方式来执行自己的代码,以达到隐藏自身的目的.

3 系统的设计和实现

为了让自己不被发现,攻击者必须保证自己的代码不会被反病毒软件检测到.

为了让内存扫描攻击不发现 Rootkit 的代码,除了让这些代码移出内存之外没有更好的办法. 但是,如果它们不在内存中,Rootkit 就无法正常工作了. 那么又有没有既能让 Rootkit 常驻内存中,正常完成其功能,但又不会被内存扫描工具发现的方法呢? 文中利用虚拟化的哈佛体系结构做到了这一点.

3.1 一种基于体系结构的方法

在当前的冯诺依曼体系结构下,采用平坦的寻址方式,数据和指令存放于一个地址空间中,而哈佛体系结构的特点之一是将对指令和数据的操作完全隔离开,使之位于两个不同的地址空间. 从内存扫描

器的角度来看,冯诺依曼体系结构下的内存是一个混合的空间,所有的指令和数据都存在于这个空间之内,都能够被内存扫描器看到. 但是,在哈佛内存体系结构下,由于内存扫描这一操作实际上是“读”操作,因此,内存扫描器所能看到的实际上只有“数据空间”这一块,不能看到“指令空间”内的任何内容. 如果 Rootkit 代码都存放于“指令空间”的话,那么任何内存扫描器都无法看到这些敏感的代码了. 虽然现在几乎所有用于通用计算的 CPU 都是以冯诺依曼的体系结构为基础,但可以利用当今 CPU 结构上的一些哈佛体系结构的特征,在冯诺依曼体系结构上做出一些修改,在一些特定的内存区域分离指令和数据的地址空间,实现虚拟化的哈佛内存体系结构,达到欺骗内存扫描器的目的.

3.2 系统的设计和实现

系统目标是:以最小的代价(包括最小的性能损失和最少的代码改动)保证常驻内存中的 Rootkit 代码不会被内存扫描工具发现.

在 X86 CPU 上,某个页面的线性地址 va 通过 MMU 进行地址翻译后得到的物理地址为 pa ,这个地址翻译的过程表示为:

$$\text{Mapping}_d(va) = pa$$

这个映射无论对于取值操作还是数据操作都是一样的,因为都使用操作系统提供的同一份页表, va 总会被映射到同一物理地址 pa .

而在系统中:

对于 $\forall va \in V$ (V 为 Rootkit 代码的线性地址空间). 对 va 的取指操作:

$$\text{Mapping}_i(va) = pa_{\text{exec}} \quad (1)$$

对 va 的数据操作:

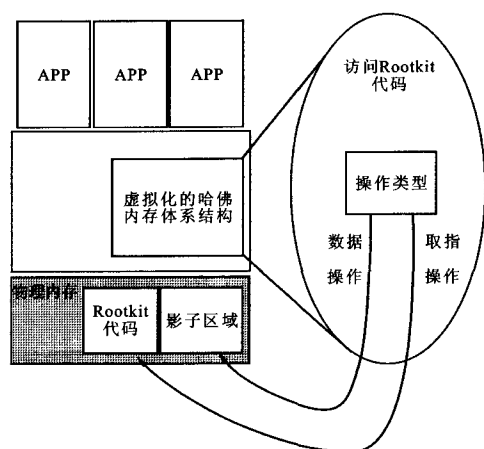
$$\text{Mapping}_d(va) = pa_{\text{data}} \quad (2)$$

其中, $pa_{\text{exec}} \in P_I$, $pa_{\text{data}} \in P_D$, 且 $P_I \cap P_D = \emptyset$. 其中 P_I 为指令物理地址空间, P_D 为数据物理地址空间, 又称为影子区域. 所有对 Rootkit 代码所在区域的读写操作都将被导向这里, 这里不会有任何引起怀疑的数据或是代码.

这种机制一旦建立起来,系统运行过程中,它将一直保证 Rootkit 代码的隐蔽性. 系统的总体结构如图 1 所示.

文中系统以 X86 上的 Linux 为基础,对地址翻译的过程进行了轻微的修改,并且利用了 X86 CPU 分离的指令和数据 TLB. 在此先简单介绍地址翻译的过程以及 X86 中分离的 TLB.

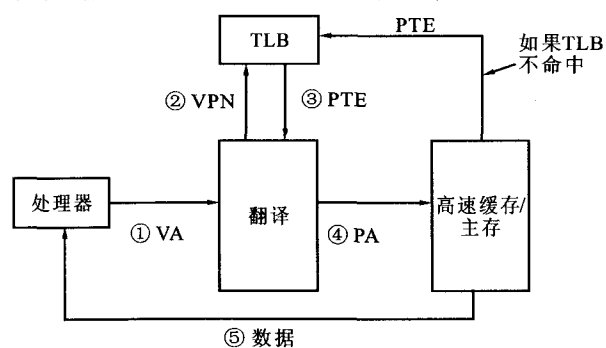
(1) X86 CPU/Linux 上的地址翻译



Linux 在 X86 上的地址翻译过程使用了多级页表,这里只讨论最后一步,即由线性地址到物理地址的转换过程。

CPU 每产生一个线性地址,MMU 就通过页表找到这个线性地址对应的物理地址,页表的每一个表项称之为 PTE(Page Table Entry)。由于页表也位于主存中,如果需要的 PTE 表项不在高速缓存中将会导致一次额外的访存操作,这个代价是非常大的。CPU 的设计人员试图消除这样的开销,他们在 MMU 中包含了一个关于 PTE 的小缓存,称为翻译后备缓存(Translation Lookaside Buffer, TLB)。

有了 TLB 以后,MMU 对线性地址和物理地址的转换速度得到了很大的提升。有 TLB 的 CPU 对某个线性地址的访问步骤如下(图 2):



第①步:CPU 产生一个线性地址。

第②步、第③步:MMU 从 TLB 中取出相应的 PTE。

第④步:MMU 从 TLB 中取出这个线性地址对应的物理地址,将它发送到高速缓存/主存。

第⑤步:高速缓存/主存将所请求的数据字返回给 CPU。

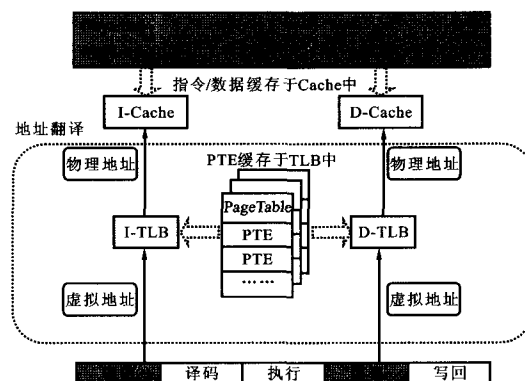
如果第②步中 TLB 不命中,MMU 在主存的页表中找到对应的物理地址进入第④步,并把相应的 PTE 取出放在 TLB 中,覆盖掉一个以及存在的条目。

可以看到 TLB 是地址翻译过程中的无法绕过的一环。

(2) I-TLB 和 D-TLB

文中充分利用 CPU 中带有哈佛体系结构思想的特征,在软件层次上实现指令和数据的分离。

在 CPU 的设计中引入了数据 Cache 和指令 Cache,以及数据 TLB(D-TLB)和指令 TLB(I-TLB)。这样 CPU 可以在取值和访存两个阶段并行地进行地址翻译和物理内存的访问。CPU 在取指阶段进行内存访问时,它会在指令 TLB 中寻找其指令的线性地址所对应的物理地址,而在访存阶段进行内存访问时,就会在数据 TLB 中寻找数据的线性地址对应的物理地址(图 3)。



(3) 系统的具体实现

虽然理论上某个线性地址的 PTE 表项在 I-TLB 和 D-TLB 中的内容都应该是一致的,但是实际上并没有相应的硬件机制来保证。利用这一点,文中找到了实现哈佛内存体系结构虚拟化的一种可行方法。能够让操作系统为 CPU 流水线的取值和访存阶段提供不同的页表,而并不需要真正地维持两份页表,只需让页表在两个 TLB 中缓存的 PTE 不相同即可。这样对 Rootkit 代码的读写和执行实际上是在两个不同的物理页面上进行,任何读写都不会影响到 Rootkit 代码的正常执行,也就实现了公式(1)和(2)中的 Mapping_i()和 Mapping_d()两个不同的地址映射方式。

具体的做法如下:

① Rootkit 安装时,在内核中分配一块与 Rootkit 代码区大小相同的连续物理内存,作为

Rootkit 代码的影子区域(P_D),并用一个只读的全局指针变量 `text_mirror(unsigned long * const text_mirror)` 指向它.把所有的 Rootkit 代码都拷贝一份放入这个区域中.

这个物理地址空间中所有页帧与 Rootkit 代码空间中的页帧一一对应. Rootkit 代码页的第一个物理页帧的页号为 `text_start`,影子区域的第一个物理页帧的页号为 `mirror_start`,内核代码中物理页号为 `ppn` 所对应的影子页的页号为 `shadow(ppn)`:

$$\text{shadow}(\text{ppn}) = \text{ppn} + (\text{mirror_start} - \text{text_start})$$

② 将所有 Rootkit 代码页所对应的 PTE 中的物理页号(ppn)部分暂时改为这些页在影子区域中对应物理页帧的页号(`shadow(ppn)`),并将其 Present 位和 Global 位(确保这些 PTE 项不会被刷出 TLB)改为 1,然后将这些 PTE 都导入到 D-TLB 中(读取某个页面上的一个字节即可让这个页面的 PTE 项被导入到 D-TLB 中).

③ 将 PTE 的物理地址字段和 Global 位的值改回原始内容(ppn),并且将 Present 标志改为 0(表示所指的页不在内存中).

④ 修改 Linux 的页错处理程序:如果是由于试图读写代码区引起的页错,就重复第②步和第③步.

经过修改后的内核启动后对 Rootkit 代码页的访问情况如图 4 所示.

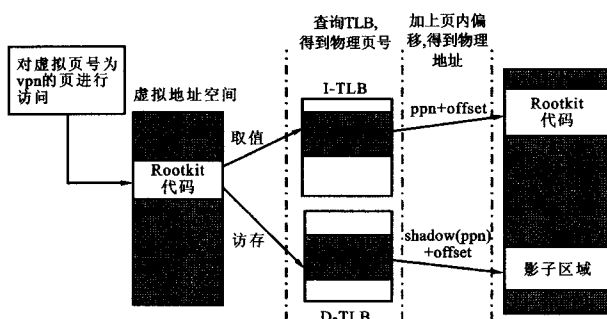


图 4 对内核代码访问时的地址翻译过程图

系统正常启动后,这种“双重”地址映射过程就已经建立.一般来说,Rootkit 代码页的 PTE 项由于设置了 Global 位,是不会被置换出去;即使被换出,由于将代码页对应的 PTE 的 Present 位改为了 0,对这一页的访问会产生页错,而在页错处理程序(第④步)中会重新建立起“双重”地址映射,就像系统刚启动时一样.

3.3 有效性分析和对操作系统的影响

(1) 有效性分析

当 Rootkit 安装以后,双重映射机制就已经建立了起来.由于查询 TLB 是 CPU 进行地址翻译的第一个环节,因此所有可能的读写操作都会被引导到 `text_mirror` 指向的影子区域中.即使代码页对应的 PTE 被换出了 TLB,由于对代码页的读写操作会引发异常,在异常处理程序中,我们将修改后的 PTE 再次导入到 D-TLB 中,异常处理程序返回之后,双重映射机制将再次建立起来.因此,机制一直存在于系统中,任何内存扫描器都无法绕过它.

(2) 对 Rootkit 运行和操作系统的影响

将所有对 Rootkit 代码的读写操作都重定位到另一个区域,而正常的执行操作时不会受到任何影响,因为正常的执行都是取值操作,都是通过 I-TLB 而非 D-TLB 来进行的.通过操纵 D-TLB 中的内容将读写操作都导向影子区域是没有问题的,Rootkit 的正常工作不会受到任何影响.

文中方法充分利用了 TLB,而且将修改过的 PTE 都导入了 TLB,这些修改过的 PTE 很少被换出.即使被换出,重新建立保护机制也只需要一次页错处理即可完成.因此,在性能方面,对系统的影响不会很大.

为了重定向对 Rootkit 代码的数据操作,将这些代码页的页表项导入了 D-TLB 中,并让其不被换出,这实际上等于减少了 D-TLB 的容量.这种做法对性能的影响我们需要通过实验来观察.

4 实验分析

Rootkit 利用 Linux 的可装载模块(LKM)安装,修改系统调用的入口地址,劫持正常的系统调用.将 Rootkit 安装在内核为 Linux 2.6.11 的 Fedora Core 4 上.为了演示实验效果,在 Rootkit 中将影子区域的内容全部初始化为‘0’. Rootkit 成功安装之后,扫描操作系统内核代码区的内容,通过 `/dev/mem` 打印出它的内容,发现 Rootkit 所在的代码区内存单元的内容已经全部为‘0’字符,这说明对 Rootkit 代码的读操作被导向了影子区域,而实际上 Rootkit 是存在于系统中的,虚拟化哈佛体系结构成功地隐藏了 Rootkit 代码.

5 结束语

通过虚拟化技术在当前流行的架构下引入其他体系结构中一些有用的特征,以增强 Rootkit 的隐蔽性,这种思想对 Rootkit 技术的研究很有启发意

(下转第 136 页)

而真/假隶属度之间的差为 0.4. 因此可以确定, 肯定存在不确定性, 即它的模糊熵一定是大于 0.6 的, 从而证实本文所提出的模糊熵计算方法的正确性和有效性.

6 结束语

Vague 集^[11-12]的理论研究还在不断的深入, 对于 Vague 集模糊熵的研究更是得到了大量的关注, 许多学者也不断的提出新的 Vague 集模糊熵的度量方法. 文中在研究现有方法的基础之上, 给出了 Vague 集的一般定义以及 Vague 集模糊熵的公理化定义, 提出了一个新的 Vague 值以及 Vague 集的模糊熵的度量方法, 并结合公理化定义对该方法给予了证明, 证实其可用性. 最后通过实验数据进一步说明了该方法的有效性和合理性.

参考文献:

- [1] Zadeh L A. Fuzzy sets[J]. Information and Control, 1965, 8(3):338-357.
- [2] Gau W L, Buehrer D J. Vague sets[J]. IEEE Transactions on Systems, Man and Cybernetics, 1993, 23(2):610-614.
- [3] 马志锋, 邢汉承, 郑小妹. 不完整 Vague 决策表中的近似学习方法[J]. 计算机研究与发展, 2000, 37(9):1050-1057.

- [4] Bustince H, Burillo P. Vague sets are intuitionistic fuzzy sets[J]. Fuzzy Sets and Systems, 1996, 79(3):403-405.
- [5] 刘云生, 黄国顺. Vague 集的熵[J]. 小型微型计算机系统, 2006, 27(11):2115-2119.
- [6] 黄国顺, 刘云生. 关于 Vague 集的模糊熵[J]. 计算机工程与应用, 2005, 41(33):48-50.
- [7] Szmidt Eulalia, Kacprzyk J. Entropy for thtutionistic fuzzy sets[J]. Fuzzy sets and systems, 2001, 118(3):467-477.
- [8] 李天志, 梁家荣, 范平, 等. Vague 集的模糊熵[J]. 计算机应用研究, 2007, 24(10):93-98.
- [9] 李艳红, 迟忠先, 阎德勤. Vague 相似度量与 Vague 熵[J]. 计算机科学, 2002, 29(12):129-132.
- [10] 范平, 梁家荣, 李天志. Vague 集的新模糊熵[J]. 计算机工程与应用, 2007, 43(13):179-181.
- [11] 吴成茂. 模糊熵的修改及其在图像分割中的应用[J]. 微电子学与计算机, 2008, 25(1):25-28.
- [12] 王新程, 杨蓉蓉. 基于最大熵的 P2P 文件污染防治机制[J]. 微电子与计算机, 2009, 26(1):190-192.

作者简介:

金 兰 女, (1981-), 硕士研究生. 研究方向为数据挖掘、粗糙集理论.

杨 勇 男, (1967-), 博士, 副教授. 研究方向为粗糙集理论.

牛彩云 女, (1981-), 硕士研究生. 研究方向为数据挖掘、粗糙集理论.

(上接第 132 页)

义. 文中利用这种虚拟化的思想, 在以冯诺依曼体系结构为基础的操作系统上实现了哈佛内存体系结构, 这种方法的优点是对操作系统的修改十分轻微. 实验结果表明, 这是一种有效的防内存扫描的 Rootkit 技术.

参考文献:

- [1] 孙淑华, 马恒太, 张楠, 等. 内核级木马隐藏技术研究与实践[J]. 微电子学与计算机, 2004, 21(3):76-80.
- [2] 孙淑华, 马恒太, 卿斯汉. 内存映射型内核级木马的研究与改进[J]. 微电子学与计算机, 2004, 21(11):15-19.
- [3] Fuzen, FU rootkit[EB/OL]. (2008-01-25)[2009-03-10]. <http://www.rootkit.com/project.php?id=12>.

- [4] John G, Levine, Julian B, et al. Detecting and categorizing kernel-level rootkits to aid future detection[J]. IEEE Security & Privacy, 2006, 4(1):24-32.
- [5] Sparks S, Butler J. "Shadow walker": raising the bar for windows rootkit detection [C]// Black Hat Japan. Japan, Tokyo, 2005.
- [6] Chen S, Xu J, Sezer E C, et al. Noncontrol-data attacks are realistic threats[C]// Proc. USENIX Security Symposium. USA: Baltimore, 2005.

作者简介:

易再尧 男, (1972-), 博士研究生. 研究方向为网络与信息安全.