基于TWPos内核完整性保护

何 进，范明钰，王光卫

（电子科技大学计算机科学与工程学院 成都 611731）

【摘要】内核rookits攻击对内核的完整性构成致命威胁，因此对内核rootkits防护是内核完整性保护的重点。当前研究主要侧重于内核rootkits探测和防护，不足之处在于：1) rootkits防护存在单一保护模式；2) 内核rootkits探测只能做探测使用，即便发现内核已经受到攻击，也无能为力。鉴于这种情况，该文设计了一种内核完整性保护方法，采用安全认证保护和探测恢复两种方式(TWPos)保护操作系统，同时具备探测和防护能力，即便内核受到攻击也能进行恢复。实验表明，TWPos系统既能全面有效的防护，而且又不牺牲系统性能，并且兼容多种OS系统。

关 键 词 探测恢复复试； 内核完整性； rootkits； 安全认证保护模式； TWPos； 虚拟机管理
中图分类号 TP309 文献标志码 A doi:10.3969/j.issn.1001-0548.2015.06.017

Protection of Kernel Integrity with Two-Mode Protection Operation System

HE Jin, FAN Ming-yu, and WANG Guang-wei

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 611731)

Abstract Kernel-level rootkits attacks pose a deadly threat to kernel integrity, and kernel rootkits is currently a research focus, primarily focused on kernel-level rootkits detection and rootkits protection. However, these studies are always flawed: the rootkits protection presents a single protected mode; kernel-level rootkits detection can only do the detection use, even if the kernel has been found to be attacked, there is no method to solve. Give this situation, we design a two-mode protection operation system (TWPos), this is, a kernel-level integrity protection method along with detection and protection capability, even if the kernel is already under attack, TWPos also recoveries kernel integrity. The experiments show that TWPos is a comprehensive and effective protection system without sacrificing system performance for the price, and is compatible with a variety of OS systems.

Key words detection recovery mode; kernel integrity; rootkits; safe authentication protection mode; two-mode protection operation system; virtual machine monitoring

Rootkits是攻击者向计算机系统植入的，能够隐藏自身踪迹并保留超级用户访问权限的恶意程序。rootkits分为用户级和内核级两种[1]，本文只对内核级rootkits进行探测和防护，内核级的rootkits对整个计算机安全构成致命威胁，它不仅破坏内核任务、文件系统、网络协议栈、系统时钟、调度堆栈等OS重要部件，而且对内核进行篡改、注入、更改内核控制流程以及隐藏自身反病毒软件。rootkits对内核攻击主要通过以下途径：1) 带有rootkits的模块插入到内核；2) 内核自身安全漏洞，注入rootkits到内核；3) 通过应用安全漏洞提取最高权限，再向内核注入rootkits。

无论采用何种途径防止内核rootkits攻击，最终目的是确保内核完整性。通过研究内核rootkits注入方式、系统特征及防范措施，将这些工作分为两大类：1) 探测内核rootkits[2-4]，早期Copilot[2]采用独立PCI卡周期获取内核的内存，并分析判断内核是否被破坏；基于Copilot改进[3]进一步探测代码完整性及其数据完整性；SBCFI[4]探测控制流；OSck[5]基于不变式进行探测。所有这些都是基于探测模式，只有rootkits已经注入内核之后，才能探测发现rootkits攻击，尽管事后采取一系列措施，然而内核完整性无法得到保障。2) 阻止rootkits注入内核从而确保内核完整性[6-9,12-13]，Secvisor[6]确保整个系统运行周期内核代码完整性；Patagonix[7]探测执行文件格式确保内核完整性；NICKLE[8]认证内核代码再执行；

收稿日期：2014 – 05 – 06；修回日期：2014 – 09 – 19
基金项目：国家863项目(2009AA01Z435, 2009AA01Z 403)；国家自然科学基金(60373109, 60272091)
作者简介：何进(1977 –)，男，博士，主要从事操作系统安全、虚拟技术及云安全等方面的研究.

HookSafe[9]函数指针保护，主要是钩子函数保护。这些保护都是基于内核某一个方面进行保护，要么保证代码完整性[6-8]，或数据完整性[11,15]或控制流完整性[9]，都采用单一防护某类攻击，而缺乏统一全面保护措施。

基于上述两种类型，仅仅采用探测方式是无法保护内核完整性，即便阻止rootkits注入确保内核完整性这种方式，也存在诸多弊端，因而无法有效的保护内核完整性。鉴于此，本文采用了一种将安全防护及探测安全漏洞并进行恢复相结合的方法确保内核完整性的方案，称为TWPos方法，TWPos基于VMM[14]管理，将TWPos保护的内核划分为两种保护措施：一部分内核采用安全认证保护方式(safe authentication protection mode, SAPM)，这部分保护主要针对内核里面安全级别要求很高的部分，如控制流、内核调度等；另一部分内核采用探测恢复方式(detection recovery mode, DRM)，这部分相对上面部分安全要求低，如一些外围设备等。TWPos将保护的内核逻辑上划分为两个部分，采用SAPM方式保护的内核部分称为SAPM-K，采用DRM方式探测保护的内核部分称为DRM-K。

1 TWPos设计

定义 1 内核完整性KI：内核整个运行周期，内核代码、内核数据、内核控制流和内核栈都未被篡改或注入，称为内核完整性。

定义 2 原始内存块OMB：OMB保存内核代码、数据、堆栈的内存。保护方式来看OMB内存=DRM-K⋃SAPM-K，DRM-K⊆OMB，SAPM-K⊆OMB并且DRM-K⋂SAPM-K=∅。

定义 3 备份内存块BMB：保存DRM-K⊆OMB备份的内存，当DRM-K完整性遭到破坏，通过BMB进行恢复，确保代码、数据、控制流和栈完整性。

定义 4 映射内存块MMB：存放的内存都是从OMB通过安全认证之后的内存，GOS可以安全访存。

定义 5 安全映射表SMT：建立SAPM-K和MMB之间映射关系，当GOS任务访问SAPM-K内存时，通过SMT项映射至MMB内存，进行访问，SMT标识状态包括5种：LOCK、UNLOCK、USE、SAFE和NORMAL。LOCK状态：建立映射时的状态；UNLOCK状态：撤销映射时的状态；USE状态：GOS可以访问内存；SAFE状态：GOS可以安全访问MMB内存，NORMAL状态：GOS通用访问OMB内存。

1.1 内存关系

将TWPos加载VMM中，TWPos从Host OS(HOS)分配3部分内存：BMB、SMT和MMB，并将这3部分内存块的页表都置为非置换方式，避免这些内存被置换出去，导致系统性能降低。

GOS未加入安全系统之前，GOS访存过程：GOS虚拟地址—>VMM。将TWPos作为VMM针对GOS的一个安全部件，被保护的GOS运行在VMM之上，加入安全TWPos系统，访存过程如图1b：GOS虚拟地址—>TWPos—>VMM。说明GOS任务安全访问物理内存时，必须通过TWPos，才能访问真正物理内存，TWPos作为GOS安全访问的重要部件，通过保护GOS物理内存达到保护GOS内核KI。内存关系如图1c：将TWPos加载到OMB时，TWPos从HOS分配3部分内存分别为BMB、SMT和MMB，处理它们之间关系如下：

1) 映射关系(OMB—>MMB)：TWPos内核加载时，将采用SAPM机制的内存或者模块SAPM-K⊆OMB内存认证备份至MMB内存SAPM-K—>MMB，并且SMT=｛OMB，MMB，SMT标识｝将它们映射关系写入空闲SMT项。

2) 副本关系(OMB—>BMB)：TWPos内核加载时，将采用DRM机制的内存或者模块DRM-K⊆OMB内存备份至BMB内存DRM-K—>BMB。

3) 安全访问(GOS—>SMT—>MMB)：若GOS访问SAPM-K内存，先查找SAPM-K内存与MMB映射关系SMT，若找到则通过SMT映射块访问MMB，由于MMB是认证之后的内存，因此该访问为安全访问。

4) 通用访问(GOS—>SMT—>OMB)：GOS访问DRM-K内存，访问过程为GOS—>SMT—>OMB。

5) 恢复关系(BMB—>OMB)：如果DRM机制周期检测到DRM-K内存被篡改，则通过BMB进行恢复BMB—>OMB。

1.2 保护机制

TWPos根据不同保护内容采用不同策略，图1显示了TWPos保护KI完整性的总体构架。

SAPM方式：初始化模块建立GOS的映射关系SAPM-K—>MMB，并将MMB内存置为只读模式，同时修改shadow page tables(SPT)页表将原先指向SAPM-K修改为指向MMB；当GOS访问SAPM-K内存通过SPT和SMT映射到MMB进行访问。要是GOS访问DRM-K内存通用访问。

DRM方式：初始化时将GOS的DRM-K内存和BMB内存切分多个逻辑块，并将DRM-K所有逻辑块与BMB建立副本关系，同时DRM-K所有逻辑块进行

HASH，并将HASH值存入BMB内存；GOS访问DRM-K内存采用通用访问，该访存未进行任何安全保护，GOS可能对DRM-K内存进行篡改和破坏。DRM启动内核线程周期HASH验证DRM-K逻辑块KI完整性，如果KI完整性被破坏则通过BMB进行恢复。

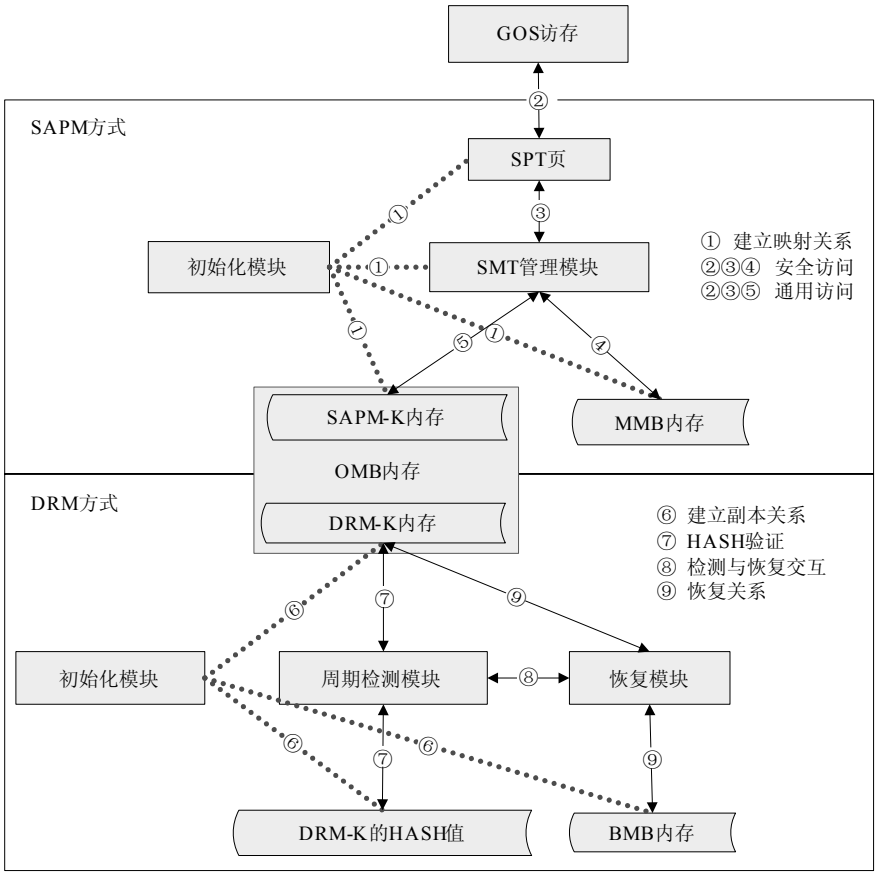

图1 TWPos总体构架

2 TWPos实现

TWPos实现两种内核和内核模块保护机制：1) SAPM方式，将内核及其模块建立映射关系SAPM-K—>MMB，GOS采用安全访问GOS—>SMT—>MMB进行访存。2) DRM方式，将DRM-K内核划分多个逻辑块，每个逻辑块DRM-K[*i*]进行HASH，并将HASH值存入OMB[*i*]内存，同时建立副本关系DRM-K—>BMB。DRM方式周期HASH验正DRM-K内核，如果探测到DRM-K内核的KI被破坏，则探测恢复关系BMB—>OMB。TWPos保护的内核，即可以采用其中一种，也可以同时采用两种保护方式，但是不能将相同内存块同时置于这两种保护方式之下。

2.1 初始化

内核和内核模块加载OMB内存中，SAPM方式保护的内核SAPM-K⊆OMB，DRM方式保护的内核DRM-K⊆OMB，并且DRM-K∩SAPM-K=∅，也就是OMB内存对应的逻辑块只能选择一种方式保护，不能将OMB的逻辑块即置入SAPM方式和DRM方式之下。从前面所述内存关系可以看出要保护SAPM-K内核，需要在初始化时建立映射关系SAPM-K—>MMB。同样，保护DRM-K内核，初始化时建立副本关系DRM-K—>BMB，并且建立HASH验证机制。同时SMT、MMB和BMB 3部分内存页表都置为非置换方式，避免内存页置换出去，导致系统性能降低。内核采用保护方式不一样，分为两种初始化过程：

1) SAPM-K初始化建立SAPM-K—>MMB映射关系，逐一将SAPM-K[*i*]—>MMB[*i*]逻辑块建立映射关系。同时建立SMT[*i*]={OMB[*i*]，MMB[*i*]，标识}映射表。

2) DRM-K内核初始化：GOS的DRM-K内核划分多个DRM-K[*i*]逻辑块并进行HASH,将HASH值—>BMB备份内存，作为后期判断DRM-K内存KI破坏的依据，建立副本关系DRM-K—>BMB，BMB为KI

内存，作为后续DRM-K内存完整性破坏时进行恢复使用。

2.2　SAPM方式

如图2所示，GOS通过SAPM方式访问内存分为两种情况：一种安全访问；一种通用访问。通用访问就是找到GOS对应的SMT项，则遵循原有内存访问模式GOS—>SPT—>SAPM-K，访存流程不变。而安全访问过程GOS—>SPT—>SMT—>MMB。GOS访问物理内存，通过SPT页表，查找访问SMT；判断SMT[*i*].标识状态，若SMT[*i*].标识=LOCK，则等待解锁之后才访存；若SMT[*i*].标识=USE&SAFE，则表示安全访问MMB[*i*]内存；若SMT[*i*].标识=USE& NORMAL，则表示通用访问OMB[*i*]内存。

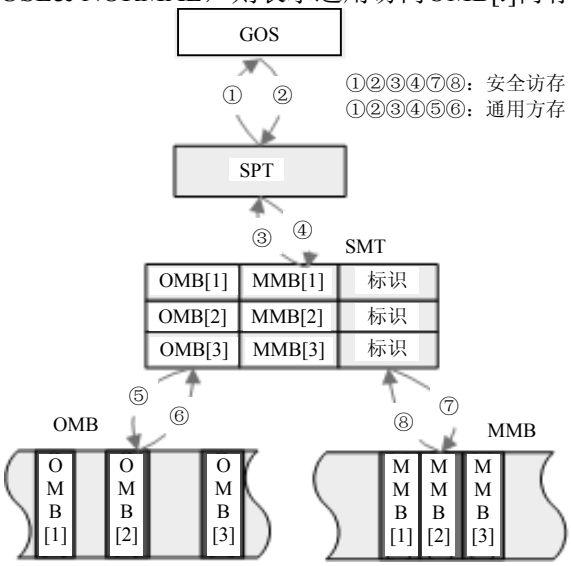

图2　SAPM保护方式下的内存访问

算法1：SAPM方式保护KI算法

页表查找GOS—>SPT.

页表访存SPT—>SMT.

if　SMT[*i*].标识 = LOCK　then

　　WAIT.

else if (SMT[*i*].标识 = USE) then

　　if(SMT[*i*].标识 = SAFE)then

　　获取SMT[*i*].MMB[*i*]地址.

　　安全访问MMB[*i*]内存.

　　else if(SMT[*i*].标识 = NORMAL)then

　　获取SMT[*i*]. OMB[*i*]地址.

　　通用访问OMB[*i*]内存.

else

　　错误

2.3　DRM方式

GOS不通过认证直接访问DRM-K内核，带有恶意代码或数据直接访问OMB，导致KI被破坏。DRM内核初始化及DRM内核模块加载过程中，DRM-K内核的OMB内存都做恢复备份和HASH。TWPos的DRM保护方式下，探测到rootkits攻击进行恢复内核线程对OMB内存做周期性校验，如图3所示，检查KI是否被破坏，若KI未破坏，则进入下一个周期检查，否则BMB内存进行恢复(见恢复算法2)。

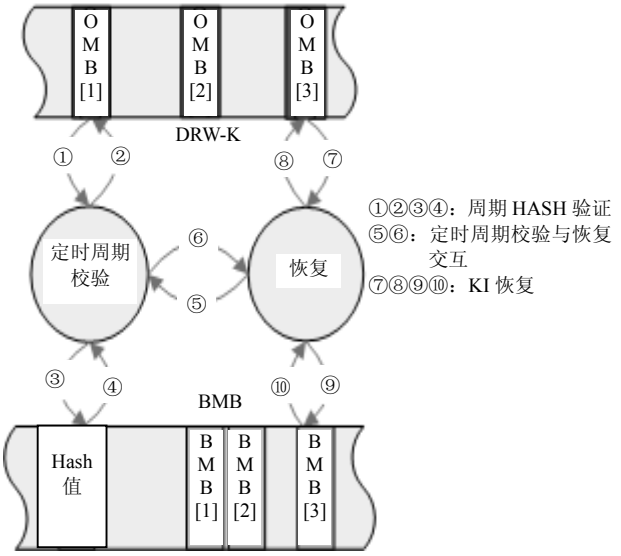

图3　DRM保护方式下，探测到rootkits攻击进行恢复

DRM方式恢复逻辑块OMB[*i*]算法如下：

1) 周期HASH验证，TWPos的内核线程对OMB[*i*]逻辑块进行HASH，值标记为：HASH[dest]。从BMB[*i*]内存获取OMB[*i*]初始化HASH值，标记为：$HASH_{[Orig]}$。$HASH_{[Orig]}$和$HASH_{[dest]}$进行匹配，若匹配则表明OMB[*i*]的KI未破坏，若不匹配则表明OMB[*i*]的KI被破坏。

2) 定时周期校验与恢复交互，定时周期校验模块发现KI被破坏，则向恢复模块发送恢复指令标记为$CMD_{[rec]}$，等到恢复模块恢复KI结束之后，向定时周期校验模块发送恢复完成指令，标记为$CMD_{[resp]}$。

3) KI恢复，恢复模块收到$CMD_{[rec]}$指令，读取BMB[*i*]，执行探测恢复BMB[*i*]—>OMB[*i*]。

算法2：恢复算法

创建周期检测线程。

for　OMB[*i*]且$\forall i \in \{1,2,\cdots,n\}$ do

　　HASH逻辑块OMB[*i*]—>HASH[dest]。

　　获取BMB[*i*]初始化值HASH[Orig]。

　　if　HASH[Orig] ≠ HASH[dest] then

　　CMD[rec]—>恢复模块.

　　LOCK—> OMB[*i*].

　　恢复关系BMB[*i*]—>OMB[*i*] .

UNLOCK—> OMB[*i*].

CMD[resp]—>定时周期检测模块.

else

OMB[*i*]的KI未破坏.

等待下一个检测周期

3 实验及其分析

TWPos系统实验及其分析侧重两种方面：1) 防护能力；2) 系统性能。实验环境：Dell PowerEdge T310、2.4 G主频、Intel Xeon X3430、4 GB内存，Xen[16] hypervisor基于3.4.2版本，dom0系统为Fedora 12，使用64位Ubuntu、内核为2.6.24作为客户机OS。

防护能力：采用现有内核rootkits攻击工具[17-23]对插入TWPos模块的系统进行攻击，测试结果如表1所示。插入模块修改控制流程、全局变量、修改或注入调用表或中断表等，从而达到隐藏或篡改操作系统目的。实验表明TWPos能够有效防止现有内核rootkits攻击，保护内核的完整性。表2通过几种探测和防护系统的比较，可看出TWPos系统是一个全面综合防御系统，包括对控制流、代码、数据完整性进行全面保护，克服单一防护模式的弱点，其他探测与防御系统对控制流、代码、数据完整性行只能保护其中一种或两种，不能全面有效的防护。

表1 通过现有内核rootkits攻击TWPos系统，防护情况

Rootkits	攻击途径	攻击对象	TWPos结果	
			输出结果	原因
Adore-ng[17]	LKM	proc_root_inode_operations->lookup	隐藏失败	HOOK重定向
		proc_root_operations ->readdir	隐藏失败	HOOK重定向
		ext3_dir_operations ->readdir	隐藏失败	HOOK重定向
		ext3_file_operations ->write	隐藏失败	HOOK重定向
		unix_dgram_ops ->recvmsg	隐藏失败	HOOK重定向
eNYeLKM[18]	LKM	Kernel代码修改	隐藏失败	内存保护
sk2rc2[19]	/dev/kmem	sys_call_table[__NR_oldolduname]	隐藏失败	内存保护
Superkit[20]	/dev/kmem	sys_call_table[__NR_oldolduname]	隐藏失败	内存保护
mood-nt[21]	/dev/kmem	sys_call_table[__NR_oldolduname]	隐藏失败	内存保护
Override[22]	LKM	sys_call_table[__NR_getuid]	隐藏失败	HOOK重定向
		sys_call_table[__NR_geteuid]	隐藏失败	HOOK重定向
		sys_call_table[__NR_getdents64]	隐藏失败	HOOK重定向
		sys_call_table[__NR_chdir]	隐藏失败	HOOK重定向
		sys_call_table[__NR_read]	隐藏失败	HOOK重定向
Sebek 3.2.0b[23]	LKM	sys_call_table[__NR_read]	隐藏失败	内存保护
hideme.vfs	LKM	Kernel代码修改	隐藏失败	内存保护

表2 TWPos与其他内核完整性探测和防护系统的对比

现有完整性保护措施	TWPos	NICKLE	SecVisor	SBCFI	HookSafe	OSck
控制流完整性	√	√	×	√	√	×
代码完整性	√	√	√	√	√	√
数据完整性	√	×	√	×	×	√

性能影响：采用XEN作为性能测试基准，Lmbench[10]作为测试性能工具，测试并分析系统各参数及TWPos系统对性能的影响。

SAPM方式和DRM方式性能对比：DRM方式获得最佳性能：最合理逻辑块切分粒度4K和最优检测周期15 ms，与SAPM方式比较，内核比例分配：SAPM-K内核占有60%，DRM-K内核占有40%，图4表明了SAPM方式比DRM方式性能更好，只要SAPM方式能够防护，尽可能采用SAPM方式。TWPos系统整体性能介于SAPM方式和DRM方式之间。

TWPos系统对应用程序的影响：应用程序访存过程：应用程序—>系统调用—>GOS内核虚拟地址—>GOS内核物理地址—>TWPos—>VMM—>物理内存。尽管TWPos只对内核完整性进行保护，但TWPos在两个方面可能对应用程序性能造成影响：1) 应用程序的系统调用需要访问内核，那么就要访问TWPos安全模块；2) DRM方式的内核线程周期检测DRM-K内存，对系统性能造成影响。图5表明TWPos系统对应用程序性能影响很小。主要基于如下原因：1) 由于TWPos采用了内存映射而摒弃仲裁技术，内存映射机制对系统性能产生轻微的影响，

不会导致系统性能严重下降；2) DRM的内核检测线程尽管周期扫描DRW-K内存，但是只有在内核被篡改或注入代码时，内核完整性遭到破坏进行恢复才会导致系统性能下降，其它时候对系统性能不造成影响。

TWPos系统对内核程序的影响：图6呈现了TWPos对系统性能的影响，导致系统性能下降的主要原因是访存过程中建立和查找对应的SMT映射。Fork进程和Exec任务都需要建立栈对象，对栈进行映射处理。Insmod插入模块对系统性能的影响主要是SMT映射的建立，并做内存备份。file copy(FC)开销主要是SMT映射函数地址的查找，而实际文件内容访问(读写)并没有增加系统开销。Apache主要针对协议栈代码完整性保护导致系统开销。TWPos对系统额外开销很少，主要开销在于SMT映射。TWPos系统对性能的影响主要是SMT导致的，访问内容不会降低系统性能。图6表明了TWPos系统比SecVisor和NICKLE系统性能好，无论系统接口调用、内核模块加载还是内核程序执行，TWPos都体现出优越的性能，主要由于TWPos摒弃仲裁技术而采用内存映射机制，并且将OMB内存切分多个逻辑块进行快速恢复。

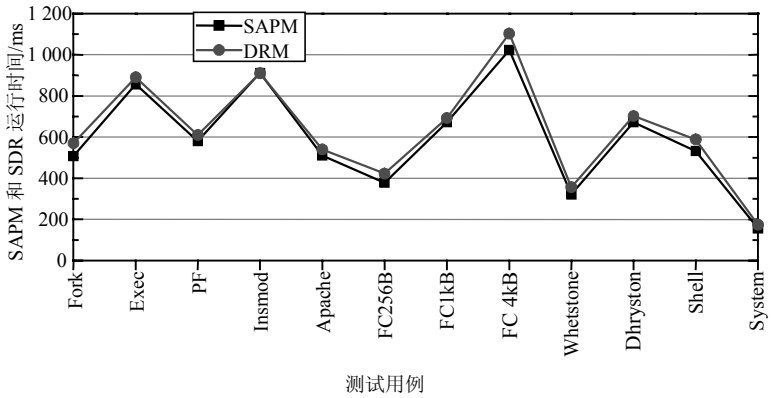

图4　SAPM与DRM性能对比

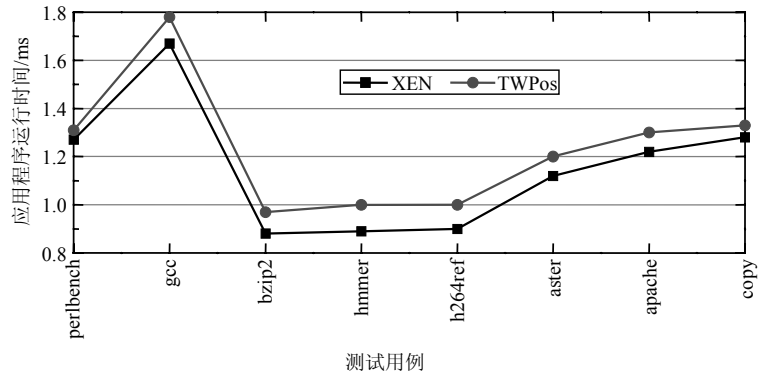

图5　TWPos和XEN性能对比

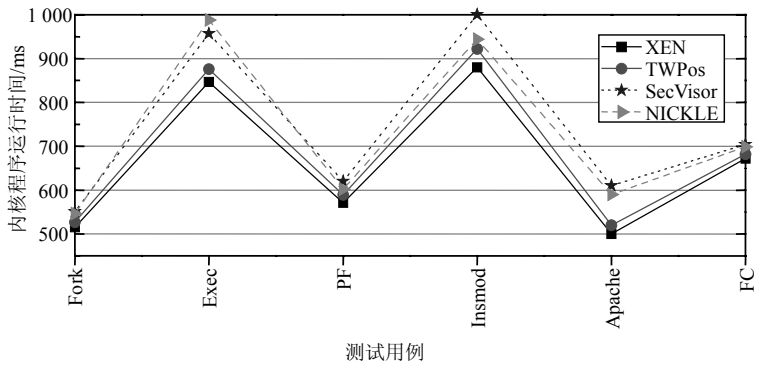

图6　TWPos与XEN、SecVisor和NICKLE内核程序性能对比

4 结 论

本论文基于VMM虚拟技术，呈现TWPos的设计、实现及其TWPos实验分析，表明了TWPos不仅能够全面高效自动探测和防护确保内核完整性，同时兼容多操作系统。因此TWPos可以广泛应用于商业领域。

参 考 文 献

[1] NGUYEN A Q, YOSHIYASU T. Towards a tamper resistant kernel rootkit detector[C]//Proceedings of the 2007 ACM Symposium on Applied Computing. Seoul, Korea: ACM, 2007.

[2] PETRONI N, FRASER T, MOLINA J, et al. Copilot: a coprocessor-based kernel runtime integrity monitor[C]// Proceedings of the 13th USENIX Security Symposium. San Diego, USA: Springer, 2004.

[3] PETRONI J N L, FRASER T, WALTERS A, et al. An architecture for specification-based detection of semantic integrity violations in kernel dynamic data[C]//Proceedings of the 15th USENIX Security Symposium. Vancouver, Canada: Springer, 2006.

[4] PETRONI J N L, HICKS M. Automated detection of persistent kernel control-flow attacks[C]//Proceedings of the 2007 ACM Conference on Computer and Communications Security. Alexandria, USA: ACM, 2007.

[5] HOFMANN O S, DUNN A M, KIM S, et al. Ensuring operating system kernel integrity with osck[J]. ACM SIGPLAN Notices, 2011, 46(3): 279-290.

[6] SESHADRI A, LUK M, QU Ning, et al. Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses[J]. ACM SIGOPS Operating Systems Review, 2007, 41(6): 335-350.

[7] LITTY L, LAGAR-CAVILLA H A, LIE D. Hypervisor support for identifying covertly executing binaries[C]// Proceedings of the 17th USENIX Security Symposium. California, USA: Springer, 2008: 243-258.

[8] RILEY R, JIANG Xu-xian, XU Dong-yan. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing[C]//the 11th International Symposium on Recent Advances in Intrusion Detection. Cambridge, USA: Springer, 2008.

[9] WANG Zhi, JIANG Xu-xian, CUI Wei-dong, et al. Countering kernel rootkits with light weight hook protection [C]//16th ACM Conference on Computer and Communications Security. New York, USA: ACM, 2009.

[10] Sourceforge site. Lmbench[R/OL]. [2014-03-19]. http:// sourceforge.net/ projects/lmbench/.

[11] LI Jin-ku, WANG Zhi, BLETSCH T, et al. Comprehensive and efficient protection of kernel control data[J]. IEEE Transactions Information Forensics and Security, 2011, 6(4): 1404-1417.

[12] SZEFER J, LEE R B. Architectural support for hypervisor-secure virtualization[J]. ACM SIGARCH Computer Architecture News, 2012, 40(1): 437-450.

[13] WANG Zhi, WU C, GRACE M, et al. Isolating commodity hosted hypervisors with HyperLock[C]//ACM European Conference on Computer Systems. New York, USA: ACM, 2012.

[14] DANIEL P. Berrangé. Virtual machine manager[R/OL]. [2014-03-29]. http://virt-manager.org/.

[15] BALIGA A, GANAPATHY V, IFTODE L. Automatic inference and enforcement of kernel data structure invariants[C]//the 2008 Annual Computer Security Applications Conference. Washington, USA: IEEE, 2008.

[16] Xen Project. Xen[R/OL]. [2014-04-19]. http://www. xen.org/.

[17] OpenWall Organization. Adore-ng[R/OL]. [2014-04-19]. http://stealth. openwall.net/rootkits/.

[18] Fr33project. Enyelkm[R/OL]. [2014-04-19]. http://www. Fr33project. org/pages/projects/enyelkm.htm.

[19] Codeforge webset. Sk2rc2[R/OL]. [2014-04-19]. http:// www.codeforge.com/ s/3/sk2rc2-code.

[20] Superkit international company. Superkit[R/OL]. [2014-04-19]. http://www.superkit.com/.

[21] Open code lib. Mood-nt[R/OL]. [2014-04-19]. http:// darkangel.antifork. org/codes/.

[22] Moodledoc site. Override[R/OL]. [2014-04-19]. http:// docs.moodle.org/23/en/Override_permissions.

[23] Sebek project site. Sebek[R/OL]. [2014-04-19]. https:// projects.honeynet.org/sebek.

编 辑 叶 芳