虚拟化环境下恶意软件攻击的修复机制

孙显军[1,2],林 闯[1],蒋屹新[1],刘卫东[1]
(1.清华大学计算机科学与技术系,北京 100084; 2.中国人民解放军 61741 部队,北京 100081)

摘 要: 恶意软件常常能够成功攻击虚拟机和其管理系统,使虚拟环境处于一种不安全、难以恢复的状态.传统的安全防护机制无法满足虚拟环境的安全要求,本文提出一种基于代理的检测和协作修复机制,通过多个虚拟机节点共享修复情况信息,快速获取有效的修复工具,提高恢复能力.模拟分析和仿真实验结果证明该机制的实用性和效率.

关键词: 虚拟化系统; 恶意软件; 攻击; 修复机制
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2011) 02-0309-06

A Recovery Scheme Against Malware Attacks in Virtualization System

SUN Xian-jun[1,2], LIN Chuang[1], JIANG Yi-xin[1], LIU Wei-dong[1]
(1. *Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*;
2. 61741 *Troops, PLA, Beijing* 100081, *China*)

Abstract: Intricate malwares can result in the failure of Virtual System, and enable the system to be in an unsafe state and difficult to restore. The existing policies thwarting this extreme attack are ineffective. In this paper, based on cooperative recovery among multiple Virtual Machines and agent-based lightweight intrusion detection, an efficient recovery mechanism is proposed for Virtualization systems against malware attacks. The basic policy is to deploy an Emergency Response/Recovery (ER) agent on Virtual Machine to identify the state of the system, and cooperative security among multiple nodes is carried out so that the infected nodes can be rapidly recovered. Simulation results also demonstrate the practicality and efficiency of the proposed schemes.

Key words: virtualization system; malware; attack; recovery mechanism

1 引言

虚拟化技术在云计算、服务计算和高效用系统中的广泛应用,使得现代社会对虚拟化环境的依赖性日益增强.然而,日益增加的恶意软件对这种高度软件化的虚拟计算环境带来了严重的安全威胁.新型安全威胁表明,恶意代码不仅能够攻击和感染真实操作系统,而且能够检测虚拟化软件,并表现出与对真实系统攻击时不同的恶意行为,使得虚拟环境安全面临严峻挑战.

针对恶意软件攻击开展的学术研究逐步成为当前的研究热点之一.Qu 等[1]提出一种在线的安全检测和分析框架,通过部署在线代理软件实现安全检测机制.Grosspietsch 等[2]进一步考虑了代理程序的自身安全问题,通过在系统的处理器或内存接口中部署特殊的诊断代理,旁路检测处理器和内存的输入输出,并当发现恶意代码时接管相应的处理器和内存控制,实现修复事务.Shi 等[3]又针对多核处理器架构提出了分层安全检测框架,非对称地配置处理器内核,将高权限内核隔离出来用于网络监视,低权限内核用于运行网络服务,实现分层安全检测.Tucek 等[4]提出一种轻量级的攻击后分析和低开销的系统恢复方法,通过权衡轻量级检测点监视和重量级检测之间的关系,实现针对快速病毒的端到端防护.郑吉平等[5]提出一种在多级安全数据库(MLS/DBMS)发生隐蔽通道渗透情况下,对同谋事务和恶意事务实施检测的方法.最近,Oberheide 等[6]提出 CloudAV 概念,将反病毒的恶意代码检测和修复作为一种云服务,通过云内部的多维病毒防护来实现快速检测和恢复.

为应对日益商业化、快速化和规模化的恶意代码制造和发布,进一步发挥多维恶意代码防护能力,本文提出一种基于代理的检测和联合修复机制(Cooperate Detection and Recovery,简称 Coop-R),以提高虚拟化系统的安全防护和应急修复能力.在这种机制下,有效的修复工具成为决定能否实现快速修复系统的关键因素.为了

收稿日期:2009-12-07;修回日期:2010-02-05
基金项目:国家自然科学基金(No.60673187,No.60803123,No.60702009);国家自然科学基金重大研究计划重点项目(No.90718040)

减少获得修复恶意攻击所需工具的时间,通过在多个虚拟机之间建立合作关系,共享状态和相关信息,降低工具获取时间.同时,为分析恶意软件攻击对虚拟化系统带来的性能和可用性影响,本文利用连续时间马尔可夫链(CTMC)对恶意攻击和修复模型进行了性能和可用性分析,并用模拟试验来交叉验证 Coop-R 机制的高效性和实用性.

2 安全威胁

当前面临的主要网络攻击可分为三类:DoS 攻击、病毒/蠕虫攻击和应用层攻击.本文主要关注病毒/蠕虫攻击,有些文献也将病毒/蠕虫称为恶意软件,即恶意代码(malicious code software)程序的缩写.近年来,随着恶意软件制造的逐步商业化,其复杂性也日益增加,尤其是以 Bagle,Warezov 和 Zhelatin 等为代表的复杂恶意程序[7,8],出现了更加复杂的特征:(1)使用反恶意代码分析方法;(2)使用激活状态方法对抗恶意代码分析方法;(3)短暂时间内投递大量恶意代码;(4)能够识别虚拟化环境,实施针对虚拟化软件的攻击.例如,恶意软件能够通过 VME 的"backdoor"通信通道的 0x5658(VX)端口识别,以及中断描述表、全局描述表和位置描述表定位来检测虚拟机[9,10].

3 修复机制

3.1 合作修复机制

充分利用网络资源和智慧实现恶意代码快速分析,提供快速解决方案成为当前恶意代码检测和修复技术新趋势.本文提出一种 Coop-R 机制,通过在多虚拟机环境中部署检测代理并建立协作,快速检测恶意代码,实施恢复.假设:

(1)虚拟化环境 VME 由 VMM 管理器和 n 个虚拟机组成,表示为 $VME=\{VMM,VM_i \mid i=1,2,\cdots,n\}$;

(2)某一时刻 t_0 虚拟环境 VME 受到一种恶意软件攻击 $M_p,p\in\{1,2,\cdots,z\}$;

(3)针对一种恶意软件攻击 m_p,存在 h_p 种修复工具 $K=\{k_1^p,k_2^p,\cdots,k_{h_p}^p\}$.

如图 1 所示,Coop-R 机制采用两层架构:综合防护层(CP)和应急响应层(ER).CP 层部署在主机操作系统层,负责检测和处理各种常见病毒.ER 部署在虚拟机上负责监视其状态,并在 CP 层失效情况下的实施恶意软件检测和修复.

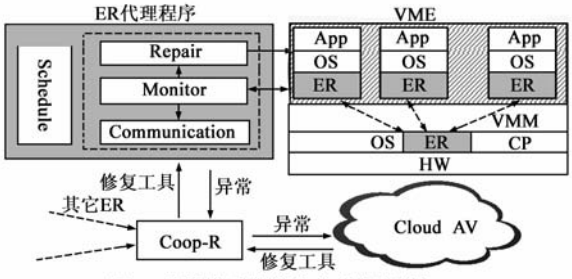


图1 虚拟化系统的协作修复机制

ER 层由四部分组成:Monitor、Communication、Repair 和 Schedule,其功能如下:

(1)Monitor 模块:监视系统状态,判断系统是否发生了恶意软件攻击.

(2)Communication 模块:负责在多个虚拟节点之间的交换信息.

(3)Repair 模块:利用修复工具执行修复事务.

(4)Schedule 模块:管理 Monitor 和 Communication 模块的执行模式,如周期性执行或事件触发执行等.

3.2 修复工具更新和维护

在 VME 中,每个虚拟机节点维护一个系统状态和修复工具列表,以便与其他虚拟机节点信息共享.维护修复工具的最新状态需要:

(1)虚拟机代理向其合作节点通告本节点状态,并当检测到恶意软件时,从本地工具列表中选择可用的最佳修复工具.

(2)完成 ER 修复事务后,对工具 k_i^p 进行量化评价,评价结果为 ω

$$\omega=\frac{\alpha(p_d+p_r)}{\tau} \tag{1}$$

其中,τ 为恶意软件检测和修复所需的时间;α 表示修复执行结束后系统恢复的可能;检测率 p_d 表示检测到的项与实际恶意软件攻击导致的变化项之比;修复率 p_r 表示已检测项中被修复的项与全部项数之比.

(3)向合作节点通告本节的状态和有关信息,更新工具评价列表.

3.3 修复过程

假设虚拟机在初始化时保存了一个工具列表(表 1),其工具选择和维护过程由算法 1 和算法 2 给出.

表 1 本地工具列表

名称	版本	修复对象	修复能力(ω)
AVG	1.1.0.4	$\{m_1,m_2,m_3\}$	$\{0.9,0.9,0.85\}$
…	…	…	…

算法 1 中:输入恶意软件名称 m_a,从本地工具列表中选择恶意软件 m_a 的最佳修复工具.

(1)选择所有可用的工具种类;

(2)从第 i 类工具的 N_i 个工具中,查询能够修复恶意软件 m_a 的工具,若查到记录,判断 ω 是否大于预定值.是,则保存记录位置;否则,继续;

(3)若循环结束,且 $u\neq 0,v\neq 0$,则返回 $K_{<u,v>}$;否

则,返回空,表示无可用工具.

算法 2 中:输入从合作节点接收的修复工具 $k_{<name,v,m,\omega>}$.

(1)计算当前本地可用的工具数 N;

(2)查找与 $k_{<name,v,m,\omega>}$ 相同的第一个记录位置 U,以及所有与 $k_{<name,v,m,\omega>}$ 类型相同的记录数 C;

(3)比较 $k_{<name,v,m,\omega>}$ 与 $k^{local}_{<name,v,m,\omega>}$ 中,所有针对恶意软件 m 的修复工具,判断其 ω 是否大于 $k^{local}_{<name,v,m,\omega>}$.是,将其代替;否则,忽略;

(4)当循环结束时,若 $i > U + C$,则说明本地列表没有与 $k_{<name,v,m,\omega>}$ 相同的工具,需要向本地列表 K 增加一项记录.

算法 1　Select tool from local list

```
Imput: K, m_a //tool list, malware
Output: k //selected tool
01: tool_selection(m_a, K){
02: a = min, u = 0, v = 0
03:   N = K.types
04:   for i = 1 to N
05:     for j = 1 to N
06:       if(Item.target = m_a)and(Item.ω > a)
07:       then K_<u,v> = K_<i,j>
08:     end for
09:   end for
10:   if(u≠0)and(v≠0)then Return K_<u,v>
11: else return Null
12: }
```

算法 2　Update local tool list

```
Input: k_<name,v,m,w>, K
Output: K̂
01: Tool_Update (k_<name,v,m,w>, K) {
/* k_<name,v,m,w>: from neighbors; K: available in local */
02:   N = K.records
03:   U = K.FindFrist(k.name), C = K.FindAll(k.name)
04:   for i = U to U + C
05: if (Item.m = k.m) and (Item.w > k.w) then
06: K_<i,j> = k_<name,v,m,w> //Replace the old one
07: if (i > U + C) /* add a new tool */
08:     then K̂ = U_{j=1}^{U+C} k ∪ k_<name,v,m,w> ∪ U_{j=U+C+2}^{N+1} k
09:   return K̂
10: }
```

4　性能分析

下面以基于主机操作系统的虚拟机为例,针对虚拟机系统无失效情况下的阻塞概率,以及攻击导致失效情况下的性能进行分析.

4.1　性能模型

大型虚拟化服务环境中,多虚拟机环境一般是采用相同的硬件和软件资源分配进行配置和部署的,因此我们假设 n 个虚拟机中,每个虚拟机的任务是速率为 λ 的泊松到达,且服务为独立的、一致的指数分布,共享 b 个可用的缓冲,所有缓冲满时到达任务将被拒绝.假设在任务期间没有故障出现,可以用 $M/M/i/b$ 队列模型来描述多虚拟机的排队系统模型.符号和描述见表 2.

表 2　符号和描述

符号	描述
λ	任务到达速率
μ	服务速率
λ_m	虚拟机感染恶意软件速率
μ_m	虚拟机恶意软件修复速率
c	虚拟机感染的覆盖率
C_i	从 i 个虚拟机重新配置到 $i-1$ 个虚拟机
R_i	从 i 个虚拟机重新启动到 $i-1$ 个虚拟机
τ	虚拟机恶意软件修复时间
δ	离开状态 C_i 的速率
β	离开状态 R_i 的速率
p_c	由 C_i 到状态 i 的概率
p_r	由 R_i 到状态 i 的概率

纯性能模型是一个如图 2 所示的同构 CTMC,状态索引表示系统中的任务数.该模型的马尔可夫链的 πP 可以通过解 $\pi PQ = 0$ 和 $\pi Pe = 1$,得

$$\pi_j^p = \frac{(\lambda/\mu)^j/j!}{\sum_{k=0}^{n}(\lambda/\mu)^k/k!} \tag{2}$$

稳定状态阻塞可能性 P_{bk} 可由上式中代替 $j = n$ 而得

$$P_{bk} = \frac{(\lambda/\mu)^n/n!}{\sum_{k=0}^{n}(\lambda/\mu)^k/k!} \tag{3}$$

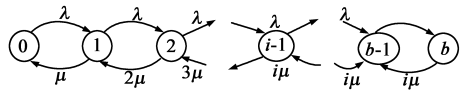


图2　性能模型

4.2　可用性模型

假设虚拟机遭受恶意软件或感染覆盖的可能性为 c,不被覆盖的可能性为 $1-c$.对于不被覆盖的感染系统可以通过简单的重新配置(如隔离)实现降级服务,而对于覆盖 VMM 的感染,则需要更长时间的修复(如清除恶意软件并重启).假设:(1)重配时间和重启时间分别服从参数为 δ 和 β 的指数分布;(2)重配和重启期间没有其他事件发生.则对应的可用性模型可以表示成一个同构 CTMC 模型,如图 3 所示.其中,状态 i($\{0,1,\cdots,n\}$,表示有 i 个虚拟机处于健康状态,而 $n-i$ 个虚拟机被感染,状态 C_i($\{C_n, C_{n-1},\cdots, C_2\}$ 表示从 i 个虚拟机重配置到 $i-1$ 个虚拟机. R_i($\{R_n, R_{n-1},\cdots, R_2\}$ 表示从 i 个虚拟机重启到 $i-1$ 个虚拟机.

解稳定状态方程可得

$$\pi_{n-i} = \frac{n!}{(n-i)!}(\rho_{m+})^i\pi_n, \quad i = 1,2,\cdots,n \tag{4}$$

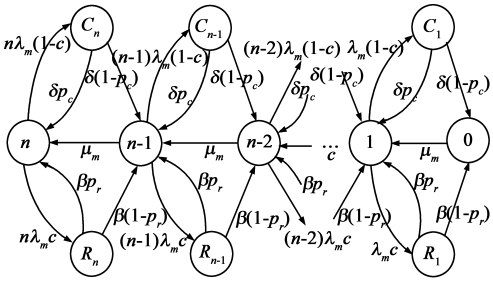


图3 复杂的失效和修复模型

$$\pi_{C_{n-i}}=\frac{n!}{(n-i)!}\frac{r(n-i)c}{\sigma}(\rho_{m+})^{i}\pi_n,\quad i=0,1,\cdots,n-2 \tag{5}$$

$$\pi_{R_{n-i}}=\frac{n!}{(n-i)!}\frac{r(n-i)(1-c)}{\sigma}(\rho_{m+})^{i}\pi_n,\ i=0,1,\cdots,n-2 \tag{6}$$

其中,

$$\rho_m=\frac{\lambda_m}{\mu_m},\rho_{m+}=\frac{\lambda_m}{\mu_m}[1-p_cc-p_r(1-c)]$$

$$\pi_n=[\sum_{i=0}^{n}(\rho_{m+})^{i}\frac{n!}{(n-i)!}+\sum_{i=0}^{n-2}(\rho_{m+})^{i}\frac{r(n-i)cn!}{\sigma(n-i)!}+\sum_{i=0}^{n-2}(\rho_{m+})^{i}\frac{r(n-i)(1-c)n!}{\beta(n-i)!}]^{-1} \tag{7}$$

将稳定状态可用性 $A(n)$定义为 n 的函数,则有

$$A(n)=\sum_{i=0}^{n-i}\pi_{n-i}=\frac{\sum_{i=0}^{n-i}\frac{(\rho_{m+})^{i}}{(n-i)!}}{\varphi}$$

$$\varphi=\sum_{i=0}^{n}\frac{(\rho_{m+})^{i}}{(n-i)!}+\sum_{i=0}^{n-2}\frac{\lambda_m(n-i)(\rho_{m+})^{i}}{(n-i)!}\{\frac{c}{\delta}+\frac{(1-c)}{\beta}\} \tag{8}$$

按照式(8),因恶意软件攻击而导致的服务不可用性为 $U(n),U(n)=1-A(n)$;而虚拟化系统 VME 在 T 时间内的不可用时间为 $D(n)=U(n)\times T$. 取 $\lambda_m=1/6000,\mu_m=1,\delta=5,\beta=12,T=8760\times60$. 从图 4($a$)可以看出感染时间对于虚拟机数是非单调递减,原因在于当虚拟机超过两个时,引起服务不可用的主要原因是重配,且重配时间随虚拟机数量线性增加. 图 4(b)表明,不可用时间 $D(n)$随 δ 的增加而降低,即减少配置时间对于虚拟化系统的服务可用性将带来明显改善.

4.3 混合模型

下面给出结合性能和可用性的 Markov 回报模型,顶层是可用性模型,底层模型是性能模型,如图 5 所示.

对于图 5 中的可用性最简单的回报分配是让系统健康状态 i 的回报速率为 $r_i=i/n$,感染状态的回报速率是 $r_i=i/n\rho,0<\rho<1$,受攻击状态的回报速率 $r_i=0$,那么面向安全的可用性回报速率的稳定状态为

$$A_s(n)=\sum_{i=0}^{n-i}r_{n-i}\pi_{n-i}=\sum_{i=0}^{n-i}(\frac{n-i}{n})\pi_{n-i} \tag{9}$$

面向能力的可用性回报速率的稳定状态为

$$A_c(n)=\sum_{i=0}^{n-i}r_{n-i}\pi_{n-i}+\sum_{i=0}^{n-i}\rho r_{n-i}C_{n-i} \tag{10}$$

因有限缓冲或服务不可用而导致的任务被拒绝的可能性 $P_{loss}(n,b)$为

$$P_{loss}(n,b)=\sum_{i=0}^{n-i}q_b(n-i)\pi_{n-i}+\sum_{i=2}^{n}C_{n-i}+\sum_{i=2}^{n}R_{n-i}+\pi_{n-i} \tag{11}$$

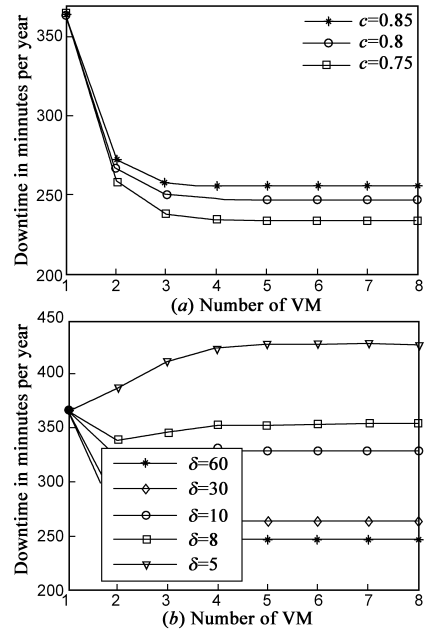


图4 参数c和δ随着虚拟机数据变化情况

取 $\lambda=40,\mu=100,\lambda_m=1/6000,\mu_m=1,c=0.85,\beta=6,\delta=5,b=10$. 图 6($a$)表明当虚拟数量大于 2 时,纯性能模型中的阻塞概率趋于0;图 6(b)描述了纯性能模型中感染对阻塞概率的影响. 系统随着虚拟机数量的增加,阻塞概率可能性增加,这是因为多虚拟机感染恶意软件可能性的增加而使每个虚拟机的任务拒绝可能性增加.

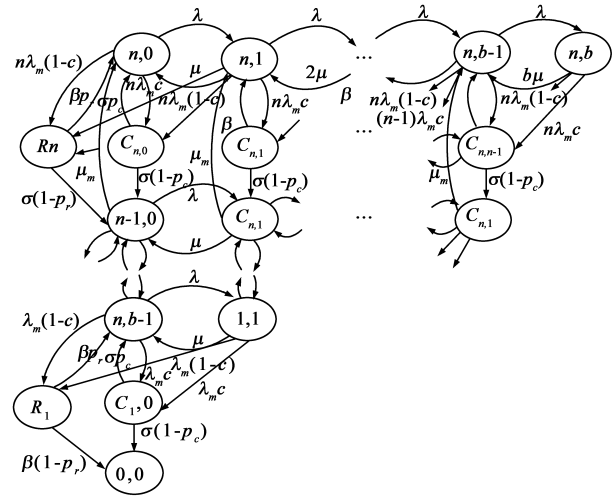
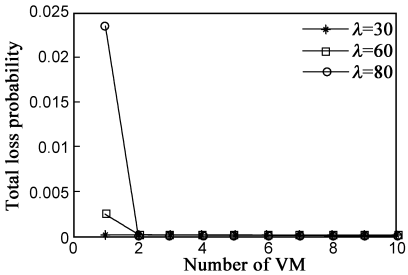
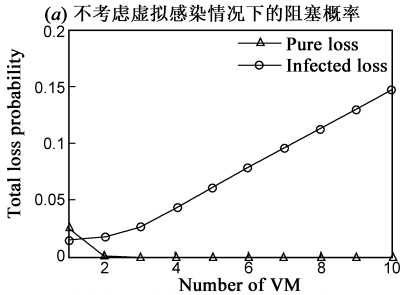


图5 性能和可用性混合模型



(*a*) 不考虑虚拟感染情况下的阻塞概率



(*b*) 感染与非感染情况下的阻塞概率对比

图6　虚拟机数量与阻塞概率关系

4.4　试验验证

为验证 Coop-R 修复机制,我们在一台 PC(Inter(R) Pentium(R) 4 2.8G CPU,1535M RAM,200 GB NTFS 格式硬盘,55%空余)上按相同硬件资源分配部署了三个虚拟机,操作系统分别为 Windows XP Home Edit, Red Hat 5.2, Windows 2003,并在此模拟环境中按下列步骤重复进行了 10 次试验.

步骤 1　初始化环境,为每个虚拟机建立快照(snapshot)和检查点(checkpoint);

步骤 2　注入病毒,运行脚本;

步骤 3　修复效果评估和数据分析.

病毒样本:Agent, Delf, Dragonbot, Feebs, Fuzen, Hacker Defender, Haxdoor, Hider, Hupigon, iBill, Kenfa, Klone, Madtol, Maslan, NsAnti, NT Illusion, NT Rootkit, Nuwar, Ramen, Morris.

rootkit: Sony rootkit (XCP/First4Internet rootkit) and Alpha DVD (Settec) rootkit.

感染速率 $\lambda_m(i,j)$取决于病毒感染能力和实际的系统环境,用模拟数据代替.修复速率与修复工具的能力有关,通过多次试验来计算平均.由表 3 可知,$\tau_{NER}=33$, $\tau_{ERI}=16$, $\tau_{ERC}=8$. τ_{NER}表示非 ER 模式下的修复速率,τ_{ERI}表示独立修复情况下的修复速率,τ_{ERC}表示 Coop-R 协作的修复速率.

表 3　三种修复方式的修复时间

方式	修复时间										平均值
NER	37	35	33	33	32	34	31	31	35	32	33
ERI	22	16	13	15	15	17	15	19	16	15	16
ERC	9	8	8	7	8	8	9	8	8	8	8

从图 7(*a*)可知,ERC 比 ERI 修复效率提高一倍.从图 7(*b*)可以看出,安全状态的可用性 $As(n)$是非递减曲线,且 $As(n,\mathrm{ERC}) > As(n,\mathrm{ERI}) > As(n,\mathrm{NER})$.

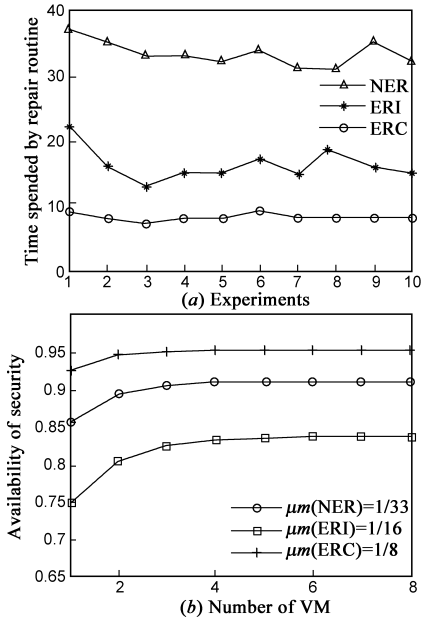


图7　不同修复机制的修复时间和可用性对比

5　结论

本文提出一种针对虚拟化系统发生恶意软件攻击情况下的合作修复 Coop-R 机制.通过多个虚拟机的安全代理之间协作响应,有效地修复极端恶意软件攻击后的系统.该机制能够针对新的恶意软件威胁,利用工具选择和更新算法快速选择有效的修复工具,提高修复效率,从而保证虚拟化系统的性能和可用性.利用 CTMC 分析了指数分布失效假设前提下虚拟化系统的可用性和性能,通过数据结果模拟分析了不同策略的修复能力对系统可用性的影响,并通过模拟试验和分析验证了该算法的有效性.下一步,我们准备在真实环境中验证和改善我们的工具选择和更新算法.

参考文献:

[1] G Qu, S Hariri, S Jangiti, J Rudraraju, S Oh, S Fayssal, G Zhang, M Parashar. Online monitoring and analysis for self-protection against network attacks[A]. Proc of Intl Conf on Autonomic Computing[C]. New York, NY, USA: IEEE CS press, 2004. 324 – 325.

[2] K E Grosspietsch, K E Silayeva. A combined safety/security approach for co-operative distributed systems[A]. Proc of the 18th international parallel and distributed processing symposium [C]. Sante Fe, NM: IEEE CS press, 2004.

[3] W Shi, H-H S Lee, L Falk, M Ghosh. An integrated framework for dependable and revivable architectures using multicore processor[A]. Proc of the 33rd International Symposium on Com-

puter Architecture[C]. Boston, MA, USA: IEEE CS press, 2006.

[4] J Tucek, J Newsome, S Lu, C Huang, S Xanthos, D rumley, D Song. Sweeper: A lightweight end-to-end system for defending against fast worms[A]. Proc of EuroSys[C]. Lisbon, Portugal: ACM Press, 2007. 115 – 128.

[5] 郑吉平,秦小麟,管致锦,孙瑾. 可生存性 MLS/DBMS 中基于隐蔽通道的恶意事务检测[J]. 电子学报, 2009, 36(6):1265 – 1269.
Zheng Jiping, Qin Xiaolin, Guan Zhijin, Sun Jin. Covert channeI based malicious transaction detection in survivable MLS/DBMS [J]. Acta Electronica Sinica, 2009, 36(6): 1264 – 1269. (in Chinese)

[6] Jon Oberheide, Evan Cooke, Farnam Jahanian. CloudAV: N-Version Antivirus in the Network Cloud[EB/OL]. http://jon.oberheide.org/files/usenix08-cloudav.pdf, 2008.

[7] Maik Morgenstern, Andreas Marx. System cleaning: Getting rid of malware from infected PCs[A]. Proc. of the 17th Virus Bulletin International Conference[C]. Ontario, Canada, 2008.

[8] S Golovanov, A Gostev, A Monastyrsky. Bootkit: the challenge of 2008[R/OL]. 18, Dec, 2008, Technique Report of Kav Lab. http://www.viruslist.com/en/analysis, 2008.

[9] Rutkowska J. Red Pill…or how to detect VMM using (almost) one CPU instruction[EB/OL]. http://invisiblethings.org/papers/redpill.html, 2004.

[10] Boris Lau, Vanja Svajcer. Measuring virtual machine detection in malware using DSD tracer[J]. Journal in Computer Cirology, 2008, 6(3): 181 – 195.

作者简介:

孙显军 男,1973 年出生于山东省单县. 清华大学计算机科学与技术系博士,研究方向为分布式系统可生存性.
E-mail: sunxj06@mails.tsinghua.edu.cn

林 闯 男,1948 年出生于辽宁省,现为清华大学计算机系教授,博士生导师,主要研究领域为 Petri 网理论、网络服务质量(QoS)控制、可信网络.
E-mail: clin@csnet1.cs.tsinghua.edu.cn